Requirements Engineering for Agile Methods

Chapter · January 2005

2 authors:

Alberto Sillitti
Innopolis University
271 PUBLICATIONS   **2,706** CITATIONS

SEE PROFILE

Giancarlo Succi
Innopolis University
540 PUBLICATIONS   **5,606** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Empirical Study on the Growth of Open Source and Commercial Software Products View project

Metrics for Energy Efficiency - Innometrics View project

14 Requirements Engineering for Agile Methods

Alberto Sillitti, Giancarlo Succi

Abstract: Collecting, understanding, and managing requirements is a critical aspect in all development methods. This is true for Agile Methods as well. In particular, several agile practices deal with requirements in order to implement them correctly and satisfy the needs of the customer. These practices focus on a continuous interaction with the customer to address the requirements evolution over time, prioritize them, and deliver the most valuable functionalities first. This chapter introduces Agile Methods as the implementation of the principles of the lean production in software development. Therefore, Agile Methods focus on continuous process improvement through the identification and the removal of waste, whatever does not add value for the customer.

14.1 Introduction

Agile Methods (AMs) are a family of software development processes that have become popular during the last few years [1, 7, 14]. Their aim is to deliver products faster, with high quality, and satisfy customer needs through the application of the principles of the lean production to software development [25].

Lean production [36] has been conceived during the '50s at Toyota [23]. It involves several practices that are now part of most manufacturing processes, such as just-in-time development, total quality management, and continuous process improvement. The principle of lean production is the constant identification and removal of waste (*muda* in Japanese), that is, anything that does not add value for the customer to the final product.

Being rooted on lean production, AMs focus on:

1. delivering value for the customer
2. ensuring that the customer understand such value and be satisfied by the project

Delivering value to the customer implies that the development team has to produce only what provides value and remove (or at least reduce to the minimum) everything else. AMs pose a lot of emphasis in producing and delivering to the customer only those features that are useful. Producing anything that is not required is considered a mistake. Adding a feature that is not needed not only con-

sumes effort without adding customer value but also creates extra code, which may contain errors and make the code longer and more complex to maintain, to correct and to improve. This waste includes general architectures that are used only partially or reusable components with functionalities that are likely to be never used [25].

To achieve such elimination of waste, AMs claim to be [7] **(a)** adaptive rather than predictive, and **(b)** people-oriented rather than process-oriented.

To ensure customer satisfaction, a close collaboration between the development team and the customer is sought, so that:

- requirements are fully identified and correctly understood
- final products reflects what the customer needs, no more and no less.

Overall, requirement engineering is of paramount importance for AMs. This chapter introduces AMs and describes their approach to requirements engineering. It is mainly related to:

- Chapter 2: most of the techniques for requirements elicitation do not change much in an agile environment.
- Chapter 4: the prioritization of requirements is of paramount importance, since AMs focus on the implementation of the most valuable features for the customer.
- Chapter 5: in order to implement only high priority features, the identification of the interaction among features and their decoupling is extremely important.
- Chapter 7: the identification of the requirements to include in a single iteration is based on the negotiation between the customer and the development team.

The chapter is organized as follows: Section 14.2 briefly introduces Agile Methods; Section 14.3 identifies common problems in requirements engineering; Section 14.4 describes the agile approach to requirements engineering; Section 14.5 deals with the role and responsibility of customers, managers, and developers in an Agile environment; Section 14.6 briefly introduces tools for requirements management in Agile Methods; Section 14.7 draws the conclusions.

14.2 Agile Methods

AMs are a family of development techniques designed to deliver products on time, on budget, and with high quality and customer satisfaction. This family includes several and very different methods. The most popular include:

- eXtreme Programming (XP) [6]
- Scrum [28]
- Dynamic Systems Development Method (DSDM) [32]
- Adaptive Software Development (ASD) [17]

- The Crystal family [12]

14.2.1 The Agile Manifesto

The promoters of AMs have realized that the wide variety of such methods may refrain potential adopters, as they could not determine what to apply in their own operations [9, 15].

As a results, such promoters have analyzed the root of lean management and have defined a document containing a set of basic values common across all AMs. Such document is called "Agile Manifesto" [7]. Being rooted in lean management, such values focus on human resources and process management:

1. **Individuals and interactions over process and tools:** the Agile approach emphasizes the importance of people and their interactions rather than focusing on structured processes and tools.
2. **Customer collaboration over contracts:** the relationship between the development team and the customer is regulated through the involvement of the customer in the development process rather than through detailed and fixed contracts (usually, contracts in agile projects are variable price-variable scope and not fixed price-fixed scope).
3. **Working software over documentation:** the goal of the development team is delivering working code, which is the artifact that provides value to the customer. Well written code is self documented and formal documentation is reduced to the minimum.
4. **Responding to change over planning:** the development team has to react quickly to requirements variability. Binding decisions affecting this ability are delayed as long as possible and the time spent in the planning activity is limited to what the customer needs. Any attempts to forecast future needs are forbidden.

From such values, a set of common practices and behaviors are identifies. The underlying claim is that they are not inventions of the Agile Community, but that they are the results of rationalizing the experience of successes and failures in software development. Some of these practices and behaviors are listed here below:

- **Adaptability:** practices have to be adapted to the specific needs of both the development team and the customer. There is no *one size fits all* solution.
- **Incremental development:** the different phases of software development (analysis, design, code, and testing) are compressed in very short iterations (from 2 weeks to 2 months) in order to focus on a few, well defined problems that provide real value to the customer (Figure 14.1).
- **Frequent releases:** at the end of every iteration, the application is released to the customer that tests it and provides feedback. This approach produces several benefits such as: **(1)** the customer can use the application very early allowing the identification of potential problems in time

for improving the product limiting the effect on the schedule; **(2)** the customer feels in control of the development process, since progresses are always visible; **(3)** the trust between the customer and the development team increases, since the team is considered reliable because it able to deliver working versions of the application early.

- **Requirements prioritization before every iteration:** before every iteration, the customer and the development team identify new requirements and reassign priorities to the old ones on the base of the customer actual needs.
- **High customer involvement**: the customer is involved in the development process through a continuous request of feedback in order to identify potential problems early in the development. In some cases, the customer is even a member of the development team (customer on site practice) and he is always available to interact with the team and clarify requirements-related issues.
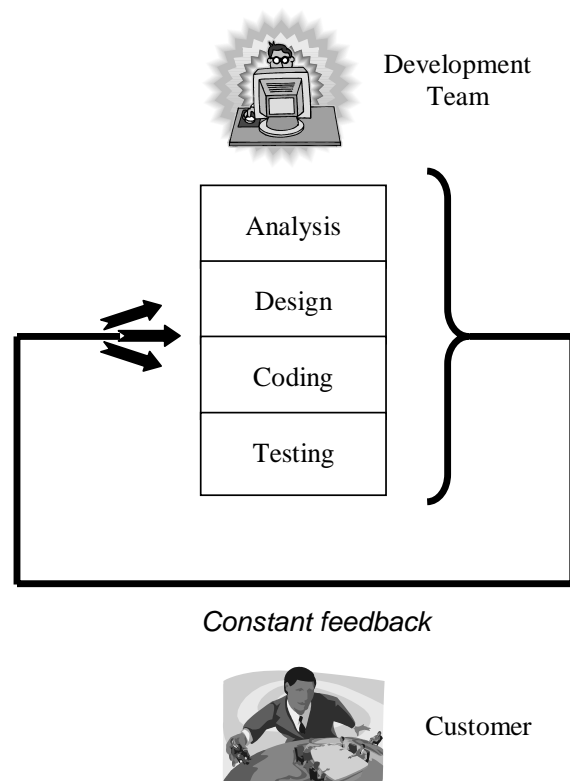

Development Team

Analysis

Design

Coding

Testing

Constant feedback

Customer

Figure 14.1: Agile development cycle

As mentioned, the basic values and practices of all the AMs are very similar. Still, by "Agile Methods" we identify a diverse family of development methodologies with different focuses and related strengths and weaknesses.

There are different levels of "agility" in AMs. A development methodology is more "agile" than another one if it requires less overhead, which is whatever does not produce value for the customer [12].

In each methodology, the development team has different priorities, processes, levels of overhead for the interaction of the team members, etc.

Therefore, there is no single solution for all the contexts. AMs provide only guidelines and a basic background of practices and behaviors that have to be adapted to the specific problem [6, 9].

The applicability of the AMs is still a matter of research [4, 34]. Issues currently being discussed include:

1. the size of the problem that can be addressed
2. how people are managed in AMs
3. the application domains in which AMs are profitable.

14.2.2 Team size in Agile Methods

Most AMs are specifically targeted to small teams, up to 16 developers (e.g., eXtreme Programming). However, there are AMs supporting a wider range of team size (e.g., the Crystal family) but there are many problems under investigation, including the use of such methods and practices in a distributed environment [14].

The level of agility is often related to the size of the development team. Direct communication and limited documentation is possible only in small teams. On the contrary, when the team grows, the level of overhead grows as well. This overhead includes: **(1)** documentation and **(2)** mediated communication. More documentation is required to share knowledge and trace the status of the project because direct, many-to-many interaction is not possible anymore [12]. Therefore, the importance of the documentation increases and it becomes a way to improve knowledge sharing. In this case, the code itself is not enough and the direct communication between the development team and the customer is not possible with a large team.

For these reasons, small teams are more agile than large teams. However, the basic principles of the lean management are still valid and most of them can scale. One of these is the continuous process improvement through the reduction of waste. This principle is useful regardless the size of the development team.

The Crystal family of AMs points out this concept [12]. Crystal includes different AMs fitting the needs of teams with different sizes (Table 14.1). The different levels of the Crystal family focus on different practices in order to manage the scalability. A limited scalability is achieved reducing the level of agility.

Table 14.1: The Crystal family

Methodology	Team (n° people)
Crystal Clear	2-6

Crystal Yellow	6-20
Crystal Orange	20-40
Crystal Red	40-80

Developing large systems using AMs is difficult or even impossible. At present, the research effort in AMs focuses on small and medium size projects, since even in this area their effectiveness is sill under investigation. Many agile practices simply do not scale, others can. AMs are adaptive [7], therefore project managers have to identify the practices to use according to the specific environment. This decision is highly affected by the size and the domain of the problem.

14.2.3 Managing People in Agile Methods

AMs focus on the value of people to solve problems and share information [11], not on the process and a massive amount of documentation [2].

However, the people-orientation can represent a main weakness for AMs since skills required to build good agile teams are not common [11].

Team members have to be excellent developers, able to work in teams, communicate and interact with colleagues and customers, etc. All these skills are required, since the team is self-organizing and cannot refer to a predefined and detailed process to solve problems and share knowledge [10].

14.2.4 Applicability of Agile Methods across Application Domains

A key question is whether AMs can be applied in all application domains. This problem is still under investigation [4, 9, 34]. In particular, how and when using specific practices results in benefits [2, 8, 27].

In general, it seems that AMs are valuable for building applications that are not mission-critical and with a limited size. Researchers are studying other areas such as the embedded systems (e.g., mobile phones and PDAs) where performances, real-time behavior, and memory constraints are common problems.

AMs focus on producing only what provides value to the customer, this does not mean that building reusable artifact such as components it is not possible. If the goal of the project is to develop a reusable artifact, the development team focuses on this problem and use AMs to address it. Reusable artifacts are not developed in projects with a different aim because developers have to include features that are not useful for the ongoing project. This approach is compliant to the principles of the AMs [7].

AMs are not the solution for developing every product. Their application is extremely hard or even impossible in many areas, such as safety-critical or very large and complex applications.

Several areas that have been analyzed in deep in traditional environments are not well understood in AMs. Often, there is a lack of research effort, especially in the area of requirements engineering [24, 34].

14.3 Traditional and Agile Requirement Engineering

Requirements are the base of all software products and their elicitation, management, and understanding are very common problems for all development methodologies. In particular, the requirements variability is a major challenge for all commercial software projects [29].

According to a study of the Standish Group [31], five of the eight main factors for project failure deal with requirements (Table 14.2): incomplete requirements, low customer involvement, unrealistic expectations, changes in the requirements, and useless requirements.

Table 14.2: Main causes of project failure

Problem	%
Incomplete requirements	13.1
Low customer involvement	12.4
Lack of resources	10.6
Unrealistic expectations	9.9
Lack of management support	9.3
Changes in the requirements	8.7
Lack of planning	8.1
Useless requirements	7.5

Engineering requirements for software systems has been perceived as one of the key steps in a successful software development endeavor, since the early days of software engineering. As a result, traditional development processes have elaborated several standards, including:

- IEEE Standard 830: Recommended Practice for Software Requirements Specifications [18]
- IEEE Standard 1233: Guide for Developing System Requirements Specifications [19]
- IEEE Standard 1362: Guide for Information Technology – System Definition – Concept of Operations Document [20]

A detailed discussion of this topic is in Chapter 8.

AMs do not rely on these standards for requirements elicitation and management but they have adapted many of the basic ideas to the new environment [3, 13, 16, 21, 24, 30, 37]. For instance, in AMs the whole development team is involved in requirements elicitation and management, while in traditional approaches often only a subset of the development team is involved.

This approach is feasible only if the size of the problem is limited. Only a small development team can interact directly with the customer. If the problem is bigger, the team can use other techniques for eliciting and managing requirements, as described in Chapters 2 and 8. This is a strong limitation of AMs.

AMs are aware that requirements variability is a constant problem in nearly all software projects; therefore, the support to such changes is included in the process as a key strength [33]. Moreover, AMs do not try to forecast changes or future needs, they focus only on the features for which the customer is paying. This approach avoids the development of a too general architecture that requires additional effort [6].

The understanding of requirements variability has a strong impact on the ability of AMs to be "lean". Often, a larger and more comprehensive architecture is expected to handle better the variability of requirements that can be forecasted in advance. However, a more complex architecture costs more not only for the development but also for the maintenance and bug fixing. Therefore, such larger architecture may end up being an inhibitor of handling the variability in requirements that cannot be forecasted in advance. Not to mention that it is usually difficult to make correct predictions, therefore many features included in the early stages of the project are not used in the final product and new ones, not identified at the beginning, are required. This approach is likely to generate useless features that are waste and generate additional waste due to the increased complexity of the code and the additional effort required to the maintenance [6, 17]. AMs focus on the development of the minimal application able to satisfy all the needs of a specific customer. Developing reusable components or framework including functionalities that are not used in the current project is considered a mistake [6].

14.4 Agile approaches to requirements engineering

AMs include practices focused on the key factors listed in Table 14.2 to reduce the risk of failure. In particular, the aim of incremental development, frequent releases, requirements prioritization before every iteration, and customer involvement is to address the main risk factors.

14.4.1 The customer

In AMs, the customer assumes a paramount role. Usually, the term "customer" identifies a set of stakeholders that belongs to the organization that is paying for the development of a software product. In this case, the interaction between the development team and the stakeholders is complex due to the different perceptions of the problem that the stakeholders have [5].

In AMs, the problem of multiple stakeholders is solved reducing their number to one, a single person that represents all the stakeholders involved in the project.

This customer should be a domain expert and able to make important decisions such as accepting the product, prioritize requirements, etc.

In the case of mass-products for which there are no organizations paying directly for the product, the development team has to identify an expert in the area (e.g., a marketing expert) that is able to act as the customer and participate in the development of the product.

This approach is feasible only if the size of the problem is limited and a single person can act as *customer*, representing all the stakeholders. If the size of the problem does not allow this approach, the team has to use other techniques to elicit and manage requirements, as described in Chapters 2 and 8.

In some AMs, the customer on site practice is common. This means that the customer is a member of the development team, is co-located with the team, and is always available to discuss issues related to the project with any team member [6].

The customer-on-site practice defines some specific requirements for the customer:

1. **Availability:** the customer has to be always available to answer questions coming from the development team. Any delay in the answer delays the development of the product.
2. **Complete knowledge:** the customer is the representative for all the stakeholders. Therefore, he is able to answer all questions, since he is the domain expert and knows how the application should work and the input/output data required. Again, this is possible if the size of the project is limited.
3. **Decision power:** the customer is able to make final decisions and commitments. Changes in requirements, acceptance of the features implemented, etc. can be decided directly by the customer, allowing a fast decision making process.

Having access to a customer able to satisfy all these requirements is not easy [26], since he has to be a very valuable member of staff. The availability of this kind of customer is of paramount importance in AMs, since most of their benefits (e.g., reduction of documentation, incremental delivery, etc.) are tightly coupled with the customer involvement [35]. However, there are attempts to extend requirements collection to involve more customers [22].

14.4.2 Waste in requirements

AMs focus on the identification and reduction of waste in the development process [25]. In particular, identifying and reducing the waste from requirements assume a paramount role to avoid the creation of waste later in the process.

In lean practices, the reduction of waste is extremely important because waste always generates further waste [23, 36]. For instance, if a factory produces more goods than required by the customers (first piece of waste) the system produces the following further waste:

- a warehouse

- people and processes to manage the warehouse
- people and processes to manage the interaction between the factory and the warehouse
- etc.

The introduction of waste in the early phases of the process causes the creation of further waste later on, the increment of the complexity, and the drain of resources available for the core business of the company. For this reasons, the optimization of a single activity produces more savings than the direct saving from the activity itself and contributes to the optimization of the whole process.

Requirements engineering in AMs focuses on [7]:

1. reduction of waste from requirements
2. managing the requirements evolution

Waste in requirements deeply affects the development process and the ability to deliver a product able to satisfy the real needs of the customer.

The main effects of waste in this area include:

- more source code to write and higher cost
- increased complexity of the source code
- delayed delivery of the final version of the application with all functionalities
- more complex and costly maintenance
- more resources required by the application, including: memory usage, processing power, network usage, etc.
- increased complexity of the application from the point of view of the customer (e.g., more complex user interface, more effort to learn how to use the application, etc.)
- savings produced by the application in the production process of the customer are delayed

At the end, all the waste generated is a cost for the customer both directly and indirectly. Such costs are likely to generate further waste inside the customer organization due to the reduced amount of money available to its core business and the reduced revenues.

Waste in requirements includes both wrong and useless requirements.

A misunderstanding between the customer and the development team causes wrong requirements. In order to reduce the probability of such misunderstanding, AMs adopt several techniques focused on the interaction between the customer and the development team:

- **The whole development team collects requirements from the customer:** requirements elicitation (Chapter 2) is an activity in which the whole team is involved. In this way, the usage of documents to share the knowledge is reduced to a minimum and the probability of misunderstandings decreases.

- **Requirements are collected using a common language:** requirements are collected using the language of the customer, not a formal language for requirements specification. This means that developers have to be introduced to the domain of the customer in order to understand him.
- **Direct interaction between the development team and the customer**: there are no intermediaries between the development team and the customer. This approach reduces both the number of documents required and the probability of misunderstanding due to unnecessary communication layers.
- **Requirements splitting:** if the development team considers a requirement too complex, this technique helps the customer to split it in simpler ones. This splitting helps developers to understand better the functionalities requested by the customer (Chapter 5).

This approach does not scale, it is feasible only if the size of the development team is limited. Otherwise, the introduction of a representative and additional documentation is required. This means that if the team size grows, some agile practices cannot be used anymore while others are still useful. In case of large projects, AMs do not provide any specific solution.

Even if the customer is an expert in its own domain, identifying the features that he really needs is not easy. Often, customers over specify the application, including a wide range of features that are not providing a real benefit for their business.

Such requirements are useless; therefore, they are a source of waste. In order to reduce this kind of waste, AMs use the following techniques:

- **Requirements prioritization:** the customer and the development team assign priorities to each requirement in order to identify more important features that have to be implemented first (Chapters 4 and 7).
- **Incremental releases:** functionalities are released in small but frequent bunches (from 2 weeks to 2 months), in order to collect feedback from the customer.

After the identification of the functionalities to include into the system, the customer and the development team assign priorities to them. The prioritization activity is performed in four steps:

1. The development team estimates the time required to implement each functionality. If the effort required is too high, the requirement is split into simpler ones that can be implemented with less effort.
2. The customer specifies business priorities for each functionality.
3. According to the business priorities, the development team assign a risk factor to the functionalities.
4. The customer and the development team identify the functionalities to implement in the iteration.

The development team and the customer repeat requirements elicitation and these four steps at the beginning of every iteration. In this way, it is possible to

identify requirements that do not provide enough value to the customer in order to discard them and focus on the most important ones.
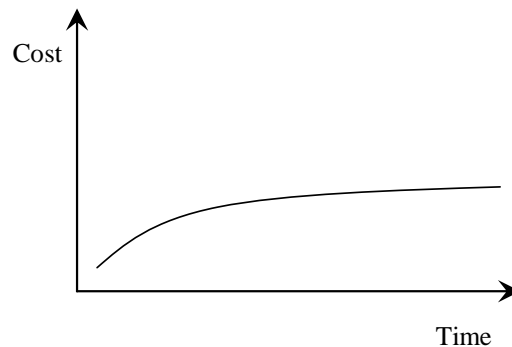
14.4.3 Requirements evolution

AMs assume that it is very hard to elicit all the requirements from the user up-front, at the beginning of a development project. They also assume that such requirements evolve in time as the customer may change its mind or the overall technical and socio-economical environment may evolve.

Therefore, Agile companies are aware that changes are inevitable and they include the management of variability into the development process.

AMs base the requirements collection and management on three main hypotheses [6]:

- requirements are not well known at the beginning of the project
- requirements change
- making changes is not expensive

In particular, AMs assume that the cost of introducing changes in a product is nearly constant over the time (Figure 14.2), but this hypothesis is not true in every context. Usually, the cost of implementing changes grows exponentially over the time. On the other hand, if development phases are grouped together in very short iterations (Figure 14.1) and binding decisions are taken as late as possible, the growing of the costs is limited [6].

Figure 14.2: Cost of changes

In order to manage requirements evolution, AMs use variable scope-variable price contracts [25]. This means that the features really implemented into the system and its cost evolve as well. Therefore, requirements are not specified in details at contract level but defined step by step during the project through a negotiation process between the customer and the development team.

Managing variability is a challenge that AMs approach in two ways:

1. **Decoupling requirements:** requirements have to be as independent as possible in order to clearly identify what to implement and make the order of their implementation irrelevant.
2. **Requirement elicitation and prioritization:** at the beginning of every iteration, there is a requirements collection and prioritization activity. During that, new requirements are identified and prioritized. This approach helps to identify the most important features inside the ongoing project. Typically, if a requirement is very important is scheduled for the implementation in the upcoming iteration, otherwise it is kept on hold. At the following iteration, the requirements on hold are evaluated and, if they are still valid, they are included in the list of the candidate requirements together with the new ones. Then, the new list is prioritized to identify the features that will be implemented. If a requirement is not important enough, it is kept on hold indefinitely.

This approach is able to identify the most important requirements during the whole project, not just at the beginning. Requirements that are not considered very important at the beginning may become relevant at some stage of the project. Moreover, the decoupling of the requirements allows the implementation of the features in nearly any order; therefore, features are implemented mainly according to their prioritization, not to their functional dependences.

14.4.4 Non-functional requirements

AMs do not provide any widely accepted technique for eliciting and managing non-functional requirements [24]. Such requirements are collected implicitly during the requirements collection activity. The need of specifying non-functional requirements is less important than in other context due to the continuous interaction with the customer. After every iteration, the product is released and the customer is able to test the product. If he identifies problems related to non-functional qualities, the team can adapt the system to meet such requirements in the subsequent iteration without affecting too much the schedule.

Often, the customer does not perceive as high impact many non-functional requirements (e.g., scalability, security, etc.). This may affect deeply the release of the final version of the application, therefore the development team has to guide the customer in order to identify such hidden needs.

This approach to non-functional requirements may represent a major risk for AMs, since they lack of specific techniques for their management.

14.5 Role and responsibility of customers, developers, and managers

AMs require a high level of interaction among customers, managers, and developers. Usually, such interaction is unmediated and all the stakeholders meet frequently in working sessions to improve the mutual understanding, the quality of the final product, and keep the project under control (on time and on budget).

Roles and responsibility of customers, managers, and developers assume a paramount importance and have a broader impact on the evolution of a software project.

14.5.1 The customer

The customer is highly involved in the development process and often he is a member of the development team. His presence is extremely important in AMs, since the amount of documentation is reduced to the minimum and the development team often asks for clarification regarding requirements.

The constant presence of the customer replaces most of the documentation required to describe requirements in details and his contribution is a key factor for the success of the project.

The customer provides feedback to the development team in order to identify potential problems early in the development and avoid a major impact on project schedule.

As stated in Section 14.4.1, the customer-on-site practice has several benefits but it is very difficult to implement. A poor implementation of this practice may reduce the effectiveness of several AMs, since many of them are tightly coupled with the involvement of the customer.

14.5.2 Developers

The whole development team is highly involved in the customer management collecting and negotiating requirements. Developers have to interact closely with the customer providing working software and collecting valuable feedback. For these reasons, the skills required by developers in agile teams are not common. They have to be very good developers, being able to work in teams, and interact with the customer using his own language [11].

Since AMs focus on this interaction, the development team has the responsibility to educate the customer. AMs require a high commitment of the customer in the project due to the frequent feedback required.

The trust between the development team and the customer assumes a paramount role. The team has to provide working and high quality software to the customer at every iteration in order to collect valuable feedback. This approach is valuable for both developers and customers. Developers can collect useful information to avoid

the implementation of useless features that increase the level of waste; customers can use (or at least test) the product after a few weeks from the project start.

14.5.3 Managers

In AMs, managers have to create and sustain a framework for the establishment of a productive interaction between the development team and the customer. They can achieve this goal identifying the best people to be included in an agile team, promoting collaboration, and negotiating contracts with the customer.

Usually, agile teams work with variable scope-variable price contracts rather than fixed price-fixed scope ones. This approach relies on the ability of the manager in the contracts definition in order to satisfy the customer and allow the maximum flexibility in the development process, as required by AMs.

14.6 Tools for requirements management in AMs

The most popular tools for requirements engineering in several AMs are paper, pencil, and a pin board.

For instance, in Extreme Programming (XP), requirements are collected through user stories. User stories are extremely short descriptions of a single functionality that the development team has to implement. They are written on small pieces of paper with the size of a postcard and hang on a pin board. The pin board is divided in three sections: user stories to be implemented, user stories under implementation, and user stories completed. This layout provides a visual representation of the project status.

Even if many Agile teams do not use computer-based tools, some of them are useful. Among these, there are standard applications not focused on AMs and ad-hoc applications developed specifically to support some agile practices.

Among the general purpose tools there are:

- **UML modeling tools:** such tools are used in two ways: **(1)** to write a high level description of the application; **(2)** to reverse engineer the code to create documentation.
- **Requirements negotiation tools:** this kind of tools helps developers and customer to identify, prioritize, and manage requirements in different environments, including the Agile one (Chapter 7).
- **Instant messaging tools:** these tools are useful to keep in touch with the customer in order to discuss requirements when he is not on-site.

Among ad-hoc applications there are:

- **Project management tools:** such tools focus on specific practices used in AMs and helps to store and retrieve requirements documents (e.g., user stories) in an electronic format.

14.7 Conclusions

This chapter has presented an introduction to the AMs and to their approaches to requirements elicitation and management. Since these methods are new, the subject is still evolving and many techniques are under investigation.

AMs seem to be a valuable approach to software development for a relevant subset of projects, but their limits are not well defined yet.

The main difference between agile and traditional methods is the involvement of the customer in the development process. Both approaches present benefits and drawbacks. In particular, AMs seem to manage effectively requirements in small projects but not in large ones. AMs focus on the production of value for the customer reducing whatever does not add value from his point of view. Therefore, the involvement of the customer is of paramount importance to achieve this goal.

On the contrary, traditional methods are able to manage effectively large project but their overhead is not suitable for smaller ones.

At present, the research in this area is very active with several papers discussed in major software engineering conferences and two specific conferences: XP200x and Agile Universe.

Acknowledgements

References

1. Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002) *Agile software development methods, review and analysis*, VTT Publications 478.
2. Ambler, S. (2001) Agile Documentation, available online at: http://www.agilemodeling.com/essays/agileDocumentation.htm
3. Ambler, S. (2002) "Lessons in Agility from Internet-Based Development", *IEEE Software*, 19(2).
4. Ambler, S. (2002) "When Does(n't) Agile Modeling Make Sense?", available online at: http://www.agilemodeling.com/essays/whenDoesAMWork.htm
5. Bailey, P., Ashworth, N., Wallace, N. (2002) "Challenges for Stakeholders in Adopting XP", *3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002)*, Alghero, Italy, 26 - 29 May
6. Beck, K. (1999) *Extreme Programming Explained: Embrace Change*, Addison-Wesley.
7. Beck, K., M. Beedle, A. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas (2001) *Manifesto for Agile Software Development*, available online at: http://www.agilemanifesto.org/

8. Cockburn, A., Williams, L. (2000) "The Costs and Benefits of Pair Programming", *1st International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2000),* Cagliari, Italy, 21 - 23 June

9. Cockburn, A. (2000) "Selecting a project's methodology", *IEEE Software*, 17(4).

10. Cockburn, A., Highsmith, J. (2001) "Agile Software Development: The Business of Innovation", *IEEE Computer*, September.

11. Cockburn, A., Highsmith, J. (2001) "Agile Software Development: The People Factor", *IEEE Computer*, November.

12. Cockburn, A. (2002) *Agile Software Development*, Addison-Wesley.

13. Duncan, R. (2001) "The Quality of Requirements in Extreme Programming", *The Journal of Defence Software Engineering*, June.

14. Cohen, D., Lindvall, M., Costa, P. (2003) *Agile Software Development*, DACS State-of-the-Art Report, available online at: http://www.dacs.dtic.mil/techs/agile/agile.pdf

15. Cohn, M., Ford, D. (2002) "Introducing an Agile Process to an Organization", available online at:
http://www.mountaingoatsoftware.com/articles/IntroducingAnAgileProcess.pdf

16. Glass, R. (2001) "Agile Versus Traditional: Make Love, Not War", *Cutter IT Journal*, December.

17. Highsmith, J.A. (1996) *Adaptive Software Development*, Dorset House Publishing.

18. IEEE Standard 830 (1998) IEEE Recommended Practice for Software Requirements.

19. IEEE Standard 1233 (1998) IEEE Guide for Developing System Requirements Specifications.

20. IEEE Standard 1362 (1998) IEEE Guide for Information Technology - System Definition - Concept of Operations Document.

21. Lee, C., Guadagno, L., Jia, X. (2003) "An Agile Approach to Capturing Requirements and Traceability", *2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, Montreal, Canada, 7 October.

22. Nawrocki, J., Jasinski, M., Walter, B., Wojciechowski, A. (2002) "Extreme Programming Modified: Embrace Requirements Engineering Practices", IEEE RE2002.

23. Ohno, T. (1988) *Toyota Production System: Beyond Large-Scale Production*, Productivity Press.

24. Paetsch, F., Eberlein, A., Maurer, F. (2003) "Requirements Engineering and Agile Software Development", *Eighth International Workshop on Enterprise Security*, Linz, Austria, 9 - 11 June.

25. Poppendieck, T., Poppendieck, M. (2003) *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison-Wesley.

26. Rasmusson, J. (2003) "Introducing XP into Greenfield Projects: Lessons Learned", *IEEE Software*, 20(3), May/June.

27. Ronkainen, J., Abrahamsson, P. (2003) "Software Development Under Stringent Hardware Constraints: Do Agile Methods Have a Chance?", *4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2003)*, Genoa, Italy, 25 - 29 May 2003.

28. Schwaber, K., Beedle, M. (2001) *Agile Software Development with Scrum*, Prentice Hall PTR.

29. Sommerville, I., Sawyer, P., (2000) *Requirements Engineering – A Good Practice Guide*, John Wiley & Sons.

30. Smith, J. (2001) "A Comparison of RUP and XP", Rational Software White Paper.

31. Standish Group, CHAOS Report 1994, available online at: http://www.standishgroup.com/sample_research/chaos_1994_1.php

32. Stapleton, J. (1995) *DSDM - Dynamic System Development Method*, Addison-Wesley.

33. Tomayko, J.E. (2002) "Engineering of Unstable Requirements Using Agile Methods", *International Conference on Time-Constrained Requirements Engineering*, Essen, Germany, 9 September.
34. Turk, D., France, R., Rumpe, B. (2002) "Limitations of Agile Software Processes", *3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002)*, Alghero, Italy, 26 - 29 May.
35. Wells, D. (2003) "Don't Solve a Problem before You Get to it", *IEEE Software*, 20(3), May/June.
36. Womack, J.P., D.T. Jones, (1998) *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*, Simon & Schuster.
37. Young, R. (2002) *Recommended requirements gathering practices*, available online at: http://www.stsc.hill.af.mil/crosstalk/2002/04/young

Author Biography

Alberto Sillitti is Assistant Professor at the Free University of Bozen, Italy. His research areas include empirical software engineering, component-based software engineering, integration and measures of web services, and agile methods.

Giancarlo Succi, Ph.D., PEng is Professor of Software Engineering and Director of the Center for Applied Software Engineering at the Free University of Bozen. His research areas include agile methods, open source development, empirical software engineering, software product lines, software reuse, software engineering over the Internet. He is author of more than 100 papers published in international conferences and journals, and of one book.