Requirements Engineering in Agile Software Development

Andrea De Lucia and Abdallah Qusef
Dipartimento di Matematica e Informatica, University of Salerno
Via Ponte don Melillo, 84084 Fisciano (SA), Italy
{adelucia, aqusef}@unisa.it

***Abstract*—Finding out, analyzing, documenting, and checking requirements are important activities in all development approaches, including agile development. This paper discusses problems concerned with the conduction of requirements engineering activities in agile software development processes and suggests some improvements to solve some challenges caused by agile requirements engineering practices in large projects, like properly handling and identifying sensitive (including non-functional) requirements, documenting and managing requirements documentation, keeping agile teams in contact with outside customers. The paper also discusses the requirements traceability problem in agile software development and the relationships between the traceability and refactoring processes and their impact on each other.**

***Index Terms*—Requirements Engineering; Agile Software Development, Traceability, Refactoring.**

I. Introduction

The agile approach is creating a stir in the software development community. Agile methods are reactions to traditional ways of developing software and acknowledge the "need for an alternative to documentation driven, heavyweight software development processes" [1]. In the implementation of traditional methods, work begins with the elicitation and documentation of a "complete" set of requirements, followed by architectural and high-level design, development, and inspection. Beginning in the 1990s, some practitioners found these initial development steps frustrating and, perhaps, impossible [2]. The industry and technology move too fast, requirements "change at rates that swamp traditional methods" [3], and customers have become increasingly unable to definitively state their needs up front while, at the same time, expecting more from their software. As a result, several consultants have independently developed methods and practices to respond to the inevitable change they were experiencing. These Agile methods are actually a collection of different techniques (or practices) that share the same values and basic principles. The Agile Manifesto states valuing " individuals and interaction over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to changes over following a plan" [1].

Requirements Engineering (RE) is the process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed. The main goal of a RE process is creating a system requirements document for knowledge sharing, while Agile Development (AD) methods focus on face-to-face communication between customers and agile teams to reach a similar goal. There are several research papers discussing the relationship between RE and AD, e.g. [4, 5, 6, 7, 8, 9]: they explain some RE practices in agile methods, compare these practices between agile and traditional development systems, and examine the problems of AD when it is dealing with the management of large projects and control critical requirements.

This paper addresses the problem of how (user) requirements can be captured and specified in the context of agile software development approaches. It therefore tries to identify how standard RE techniques and processes can be combined with agile practices and to find solutions to some of the difficulties related to their work. In addition, this article discusses the traceability problem in agile software development, since the current traceability between agile software artifacts is ill defined [10]. In particular, we discuss how to solve the traceability problem by extracting some important information from software artifacts to identify a traceability links between them, we also discuss how these links can be used to improve the decisions making process and help developers during the refactoring process. Finally, the paper comes up with a set of guidelines for agile requirements engineering.

The paper is organized as follows; the next Section sheds light on the importance of agile development in IT organizations and the benefits and limitations of agile methodologies in the software development life cycle and discusses some of agile approaches from a requirements engineering perspective. The agile RE activities are discussed in detail in Section 3, beginning with the objectives of the activity and explaining the techniques used to achieve these goals in AD, then the problems of each activity are identified and improvements to remedy these problems are discussed. In Section 4 some guidelines and enhancements are described concerned with an efficient application of RE practices in AD. Finally, Section 5 summarizes our conclusions and future work.

doi:10.4304/jetwi.2.3.212-220

II. AGILE SOFTWARE DEVELOPMENT

The goal of agile methods is to allow an organization to be agile, but what does it mean to be Agile? Jim Highsmith says that being Agile means being able to "Deliver quickly. Change quickly. Change often" [2]. While agile techniques vary in practices and emphasis, they follow the same principles behind the agile manifesto [1]:

- Working software is delivered frequently (weeks rather than months).
- Working software is the principal measure of progress.
- Customer satisfaction by rapid, continuous delivery of useful software.
- Even late changes in requirements are welcomed.
- Close daily cooperation between business people and developers.
- Face-to-face conversation is the best form of communication.
- Projects are built around motivated individuals, who should be trusted.
- Continuous attention to technical excellence and good design.
- Simplicity.
- Self-organizing teams.
- Regular adaptation to changing circumstances.

Agile development methods have been designed to solve the problem of delivering high quality software on time under constantly and rapidly changing requirements and business environment. Agile methods have a proven track record in the software and IT industries. Fig. 1 shows that about 69% of organizations are adapting one or more of agile practices for use in general project management as well as organizational development [11].

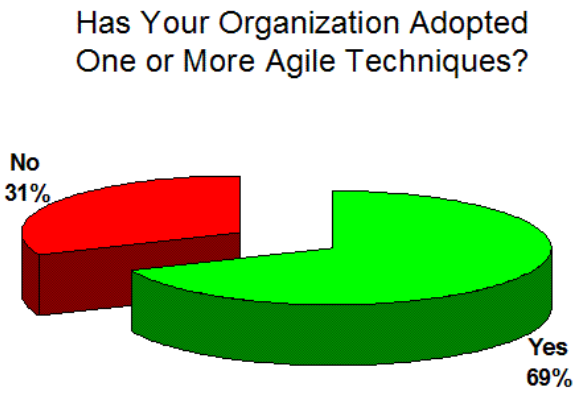

Figure 1 Agile Development Adoption

In fact, the agile development methodologies are used in organizations where there is no requirement freezing, incremental and iterative approach is used for modeling and every one in the team is an active participant and everyone's input is welcome. The main benefit of the agile development software is that it allows for an adaptive process - in which the team and development react to and handle changes in requirements and specifications, even late in the development process. Through the use of multiple working iterations, the implementation of agile methods allows the creation of quality, functional software with small teams and limited resources. The proponents of the traditional development methods criticize the agile methods for the lightweight documentation and inability to cooperate within the traditional work-flow. The main limitations of agile development are: agile works well for small to medium sized teams; also agile development methods do not scale, i.e. due to the number of iterations involved it would be difficult to understand the current project status; in addition, an agile approach requires highly motivated and skilled individuals which would not always be available, lastly, no enough written documentation in agile methods lead to information lose when the code is actually implemented. However, with proper implementation the agile methods can complement and benefit traditional development methods. Furthermore, it should be noted that traditional development methods in non-iterative fashions are susceptible to late stage design breakage, while agile methodologies effectively solve this problem by frequent incremental builds which encourage changing requirements. In the following, some common agile methods are briefly discussed from a requirements engineering perspective.

Agile Modeling (AM) is a new approach for performing modeling activities [12]. It gives the developers a guideline of how to build models - using an agile philosophy as its backbone- that resolve design problems and support documentation purposes but not 'over-build' these models. The aim is to keep the amount of models and documentation as low as possible. The RE techniques are not explicitly referred in AM but some of the AM practices support some RE techniques like brainstorming.

Feature-Driven Development (FDD) consists of a minimalist, five-step process that focuses on building and design phases [13] each defined with entry and exit criteria, building a features list, and then planning-by-feature followed by iterative design-by-feature and build-by-feature Steps. In the first phase, the overall domain model is developed by domain experts and developers. The overall model consists of class diagrams with classes, relationships, methods, and attributes. The methods express functionality and are the base for building a feature list. A feature in FDD is a client-valued function. The feature lists is prioritized by the team. The feature list is reviewed by domain members [14]. FDD proposes a weekly 30-minute meeting in which the status of the features is discussed and a report about the meeting is written.

Dynamic Systems Development Method (DSDM) was developed in the U.K. in the mid-1990s. It is an outgrowth of, and extension to, Rapid Application Development (RAD) practices [15]. The first two phases of DSDM are the feasibility study and the business study. During these two phases the base requirements are elicited. Further requirements are elicited during the

development process. DSDM does not insist on certain techniques. Thus, any RE technique can be used during the development process [9]. DSDM's nine principles include active user involvement, frequent delivery, team decision making, integrated testing throughout the project life cycle, and reversible changes in development.

Extreme Programming (XP) is based on values of simplicity, communication, feedback, and courage [16]. XP aims at enabling successful software development despite vague or constantly changing software requirements. The XP relies on the way the individual practices are collected and lined up to function with each other. Some of the main practices of XP are short iterations with small releases and rapid feedback, close customer participation, constant communication and coordination, continuous refactoring, continuous integration and testing, and pair programming [17]. Table I shows how RE activities are implemented in XP approach. In fact, XP is the most famous of any of the agile approaches.

Scrum is an empirical approach based on flexibility, adaptability and productivity [18]. The Scrum leaves open for the developers to choose the specific software development techniques, methods, and practices for the implementation process. Scrum provides a project management framework that focuses development into 30-day Sprint cycles in which a specified set of Backlog features are delivered. The core practice in Scrum is the use of daily 15-minute team meetings for coordination and integration. Scrum has been in use for nearly ten years and has been used to successfully deliver a wide range of products; Table II summarizes how RE activities are implemented actually in Scrum.

In this article some recommendations are suggested for agile development teams to help them in managing and implementing large projects and projects with critical requirements.

TABLE I.
RE IMPLEMENTATION IN XP

RE activity	XP implementation
Requirements Elicitation	• Requirements elicited as stories. • Customers write user stores.
Requirements Analysis	• Not a separate phase. • Analyze while developing. • Customer prioritizes the user stories.
Requirements Documentation	• User stories & acceptance tests as requirements documents. • Software products as persistence information. • Face-to-face communication.
Requirements Validation	• Test Driven Development (TDD). • Run acceptance tests. • Frequent feedback.
Requirements Management	• Short planning iteration. • User stories for tracking. • Refactor as needed.

TABLE II.
RE IMPLEMENTATION IN SCRUM

RE activity	Scrum implementation
Requirements Elicitation	• Product Owner formulates the Product Backlog. • Any stakeholders can participate in the Product Backlog.
Requirements Analysis	• Backlog Refinement Meeting. • Product Owner prioritizes the Product Backlog. • Product Owner analyzes the feasibility of requirements.
Requirements Documentation	• Face-to-face communication.
Requirements Validation	• Review meetings.
Requirements Management	• Sprint Planning Meeting. • Items in Product Backlog for tracking. • Change requirements are added/deleted to/from Product Backlog.

III. REQUIREMENTS ENGINEERING FROM THE AGILE DEVELOPMENT POINT OF VIEW

RE is concerned with discovering, analyzing, specifying, and documenting the requirements of the system. RE activities deserve the greatest care because the problems inserted in the system during RE phase are the most expensive to remove. As shown in Fig. 2, some studies revealed that around 37% of the problems occurred in the development of challenging systems are related to the requirements phases [19].

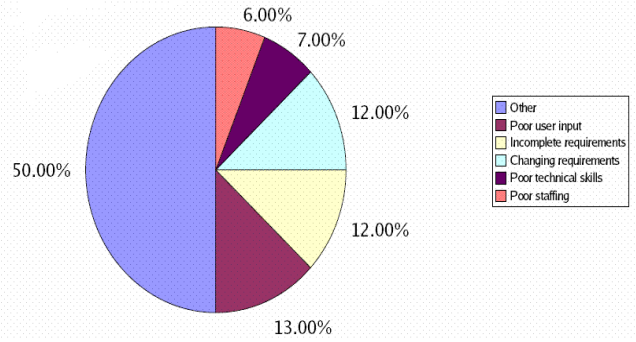

Figure 2 Problems of challenging systems

The main difference between traditional and agile development is not whether to do RE but when to do it. The RE processes in traditional systems focuses on gathering all the requirements and preparing the requirements specification document before going to the design phase, while the agile RE welcomes changing requirements even late in the development lifecycle.

Agile RE applies the focal values mentioned in the agile manifesto to the RE process. The processes used for agile RE vary widely depending on the application domain, the people involved and the organization developing the requirements. However, this paper explains the agile RE activities which are: Feasibility

Study, Elicitation and Analysis, Documentation, Validation, and Management.

A. Feasability Study

The Feasibility Study gives the overview of the target system and decides whether or not the proposed system is worthwhile. The input of the feasibility study is an outline description of the system and how it will be within an organization. The results should be a short report, which recommends whether or not it is worth carrying on with the RE and AD process. Initially, all relevant stakeholders have to be defined, in other words, all right customers who are related to the development of the system and are affected by its success or failure must be selected, and then the brainstorming session takes place to share the knowledge ideas between agile teams and "ideal" customers to answer a number of questions like:

1) **Does the system contribute to the high level objectives and the critical requirements of the organization?**
   In a first step, the high level goals and critical requirements (functional and non-functional requirements) for the system are defined upfront in order to determine the scope of the system; these requirements describe the expected business values to the customer.
2) **Is your organization ready for the AD?**
   Each agile method has its own characteristics and practices that will change the daily work of the organization. Before an organization selects one of them, it should consider whether or not it is ready for agile development. This is a very important question and many researchers tried to answer it [11, 20]. For example, Ambler [11] discusses some successful factors and questions to be answered affecting the successful adoption of agile methods.
3) **Can the system be implemented within given budget?**
   Some contracts do not allow for changing requirements. "The requirements must be complete before a contract can be made, which is often found in fixed-priced projects" [6]. In agile projects where changing requirements is welcomed, contracts often are based on time and expenses and not on fixed-priced scope. Also, "agile methods use scope-variable price contracts" [21]. 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 [8].
4) **How to integrate the agile activities with traditional organizational activities already in place?**
   Some researches suggest tentative models for integrating agile activities with traditional organizational activities by transferring the knowledge from one process to another and how the traditional team should adopt its activities to suit the mechanisms of agile teams [22, 23].

B. Requirements Elicitation

In this activity, agile teams work with stakeholders to find out about the application domain, the services that the system should provide, the system's operational constraints, and the required performance of the system (non-functional requirement). The most important techniques used for requirements elicitation in AD are:

1) **Interviews:** "Interviewing is a method for discovering facts and opinions held by potential stakeholders of the system under development" [7]. There are two types of interviews: Closed interviews, where a predefined set of questions are answered, and the Open interviews, where there is no predefined agenda and a range of issues are explored with stakeholders. In fact, interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system, but they are not good for understanding domain requirements. All agile methods say that interviews are an efficient way to communicate with customers and to increase trust between two sides.
2) **Brainstorming:** this is a group technique for generating new, useful ideas, and promoting creative thinking. Brainstorming can be used to elicit new ideas and features for the application, define what project or problem to work on and to diagnose problems in a short time. The project manager plays an important role in brainstorming. He/she determines the time of creative session, makes sure that there is no escalating discussions about certain topics, and comes to make sure that every body expresses his/her opinion freely. After the creative session is ended, the topics are evaluated by the team. Also, the connections and dependences between the discussed ideas are represented by (for example) graph visualization, so the conflicts with other requirements are found and evaluated.
3) **Ethnography:** it is an observational technique that can be used to understand social and organizational requirements [24]. In agile development ethnography is particular effective at discovering two types of requirements: the first one refers to requirements that are derived from the way in which people actually work rather than the way in which process definitions say they ought to work, and the second one refers to requirements that derived from cooperative and awareness of other people's activities. Ethnography is not a complete approach to elicitation and it should be used with other approaches such as use case analysis [19, 24].
4) **Use Case analysis:** this is a scenario based technique used in UML-based development

which identifies the actors involved in an interaction and describes the interaction itself. A set of use cases should describe possible interactions that will be presented in the system requirements; each use case represents a user-oriented view of one or more functional requirements of the system [24].

C. Requirements Analysis

The main task here is to determine whether the elicited requirements are unclear, incomplete, ambiguous or contradictory, and then resolve these issues. Conflicts in requirements are resolved through prioritization negotiation with stakeholders. The main techniques used for requirements analysis in agile approaches are:

1) **Joint Application Development (JAD):** this is a workshop used to collect business requirements while developing a system. The JAD sessions also include approaches for enhancing user participation, expediting development, and improving the quality of specifications [24]. In agile environment, in case of conflicts between stakeholders' requirements the use of JAD can help promoting the use of a professional facilitator who can help to resolve conflicts. In addition, the JAD sessions encourage customer involvement and trust in the developed system.
2) **Modeling:** system models are important bridge between the analysis and the design process [7]. In agile environment the pen board (or pin board also) is divided into three sections: models to be implemented, models under implementation, and models completed. "This layout provides a visual representation of the project status" [8]. These models must be documented and not throw-away.
3) **Prioritization:** agile methods specify that the requirements should be considered similar to a prioritized stack. The features are prioritized by the customers based on their business value, so that the agile teams estimate the time required to implement each requirement. The agile team must distinguish between "must have" requirements from "nice to have" requirements, this can be done by frequent communications with the customers.
   Fig. 3 shows the Requirements prioritization process: "at the beginning of each 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 it 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" [8].

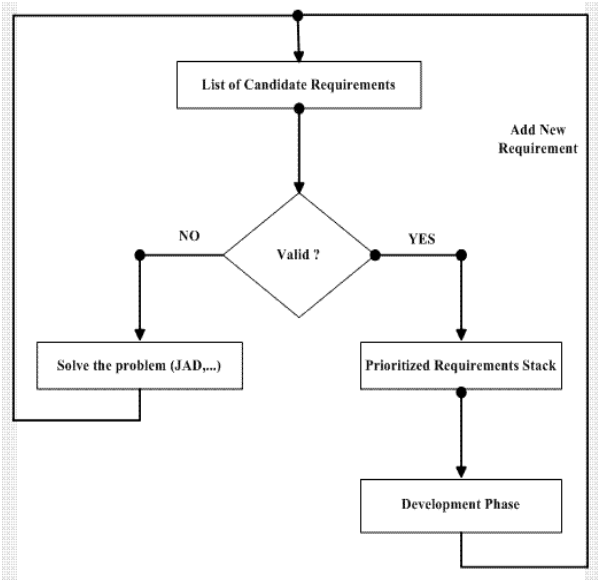

Figure 3 Requirements prioritization process

D. Requirements Documentation

The purpose of requirements documentation is to communicate requirements (or knowledge sharing) between stakeholders and agile teams. In fact, no formal requirements specification is produced in agile development methods since agile focuses on minimal documentation. The features and the requirements are recorded on story boards, index cards, and paper prototypes like use cases and data flow diagrams.

The lack of documentation might cause long-term problems for agile teams [7], so, we suggest some techniques to solve this problem:

1) The agile team leader assigns two or three members to produce documentation in parallel and concurrence with development. The two (or three) members will be responsible for handling requirements (functional and non-functional requirements), writing, reviewing, and maintaining documentation consistent with development. Furthermore, efficient practices like peer interviews will help to ensure the accuracy and quality of the documentation. The reason for choosing two or three members is because the resources are limited and the other members must adhere to the agile manifesto of producing working software rather than documentation. In addition, we can not have just one person doing it, because that violates one of the agile manifesto principles [1] "Business people and developers must work together daily throughout the project".
2) Using computer-based tools like UML modeling and project management tools to specify a high level description of the project, and to document certain practices and requirements used in agile projects in an electronic format.
3) Developing a reverse engineering process [25] to be applicable on agile projects, so that we can use it to reverse engineer the code to produce

documentation using for example UML modeling tools.

E. Requirements Validation

The goal of requirements validation is to ensure that requirements actually define the system which the customer wants. The requirements validation checks the consistency, completeness and realism of requirements. The main practices used for requirements validation in agile approaches are:

1) **Requirements reviews:** it is a manual process that involves multiple readers from both agile team and stakeholders checking the requirements against current organizational standards and organizational knowledge for anomalies and omissions. In agile projects the requirements reviews must be formal reviews: we mean that the agile team should walk with the customers through each requirement; conflicts, errors, extra, and omissions in the requirements should be formally recorded.
2) **Unit testing:** In agile, unit testing is a method for requirements validation and therefore also part of requirements engineering. In some agile methods like XP, the requirements are implemented and tested using the TDD technique. By applying this technique developers create tests before writing code. The developed code is then refactored to improve its structure [32]; the rule here is to write a code if and only if a test fails. This technique has some advantages; it is the greatest advantage to set test cases that test your requirement very accurately. The requirement from which the test case was created is now presented in a form in which it is completely validated, in the sense that it can be automatically (after each iteration) determined whether a requirement is implemented by the software or not. This makes the developers aware for the progress of the project and the state of the current iteration of the project. Also, supports the refactoring process to get an improved design by reduced coupling and strong cohesion [26]. A common misconception is that all of the tests are written prior to implementing the code [9]. Rather, TDD contains short iterations which provide rapid feedback. Code refactoring and unit tests ensure that emerging code is more simple and readable. In fact, unit tests can be considered as a live and up-to-date documentation: they represent an excellent repository for developers trying to understand the system, since they show how parts of a system are executed.
3) **Evolutionary prototyping:** a prototype is an initial version of the system. Evolutionary prototyping starts with a relatively simple system which implements the most important customer requirements which are best understood and which have the highest priority. The system prototypes allow customers to experiment to see how the system supports their work (requirements elicitation), and may reveal errors and omission in the requirements which have been proposed (requirements validation). As shown in Fig. 4, the main objective of evolutionary prototyping in AD is to deliver a working system to customers by focusing on customer interaction, [24]. The verification (Are we building the system right?) and validation (Are we building the right system?) [27] of agile projects which have been developed using evolutionary prototyping can only therefore check if the system is adequate, that is, if it is good enough for its intended purpose; in other words, verification and validation of requirements in agile systems usually rely on the validation process.
4) **Acceptance testing:** acceptance testing is a formal testing conducted by the customer to ensure the system satisfies the contractual acceptance criteria. The acceptance tests are not different than the automated system tests, but they are performed by the customer. Delivering working software to the customer is a fundamental agile principle and hence. The customers create acceptance criteria for the requirements and test the requirements against these criteria. Being AD an incremental process, the customers can give feedbacks to the developers to enhance the development of future increments of the system. However, as a general problem there are often no formal acceptance tests for non-functional requirements.

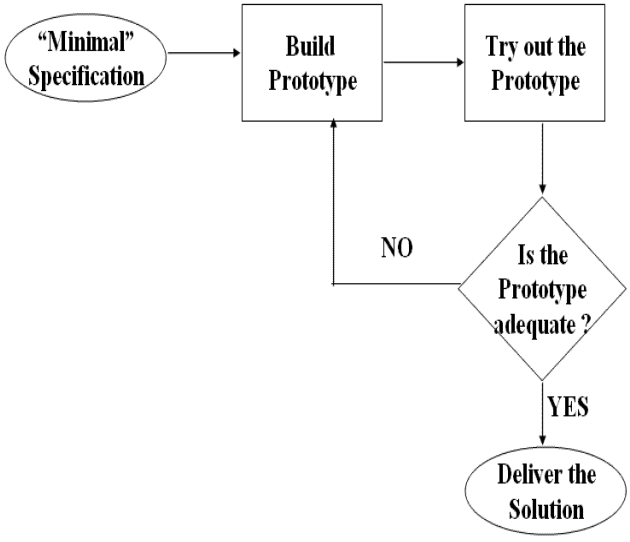

Figure 4 Evolutionary prototype processes

F. Requirements Management

Understanding and controlling changes to system requirements take place in this activity. In order for requirements management tools to work efficiently, "they must be able to store requirements, prioritize requirements, track requirement changes and development progresses, and provide a level of requirements traceability" [28, 29].

In agile projects, managers have to create and maintain a framework for the interaction between the agile teams and the stakeholders, by identifying the ideal people who

can be members of agile teams and ideal customers who can answer all the developers questions correctly [7], strengthening the collaboration, and negotiating contracts with the customers [8].

We believe that agile methods can play an important role in the management of large projects. The decomposition of the larger parts of the project into smaller components, called sub-components, lends itself to the employment of more agile teams. These agile teams can work in other time zones and other countries provided that frequent communications and self organization are established. Agile teams working in parallel on sub-components allows for quick development and an early design. An early design leads to an early review. Consequently, the iterative schedule and emphasis on delivering the product allows the agile teams to assess the successes and shortcomings, and plan for the next iteration. Once a specific agile team has successfully completed a sub-component, the team is available to work on another component or sub-component. Each of these smaller agile teams will still be responsible for assigning two members to complete the previously described documentation which is necessary to satisfy the other stakeholders.

Agile teams should use modern communications like web-based shared team projects and instant messaging tools; these tools are useful to keep in touch with the customer and other agile teams in order to discuss requirements when they are not on-site.

The ability to trace the software artifacts through the system lifecycle (source code, acceptance tests, requirements, and design logic) is critical to the success of large complex projects. Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction [30]. One of the problems is that traceability is an important part in traditional software development but it is not a standard practice for the agile methods. There are many techniques that have been presented to solve traceability issues. These techniques have been intended to work with traditional software development methodologies and therefore designed under the assumption that a formal requirements process is in place, but in agile software development the situation is different because the main development artifact in agile methods is a source code. In agile process, requirements, acceptance tests, unit tests and code change at the same time, so the unit tests should be traced to code, and the acceptance tests must include references to the requirements they test, see Fig. 5 [31].

As we say before, the main software development practice used in agile is TDD. The key aspect of TDD is that it can be viewed as a source of free traceability information. In turn, if such information is available to the developer, it may improve the efficiency with which tests are produced and code is written for each iteration. In TDD a traceability matrix is obtainable by matching new tests with changes in the code [31].

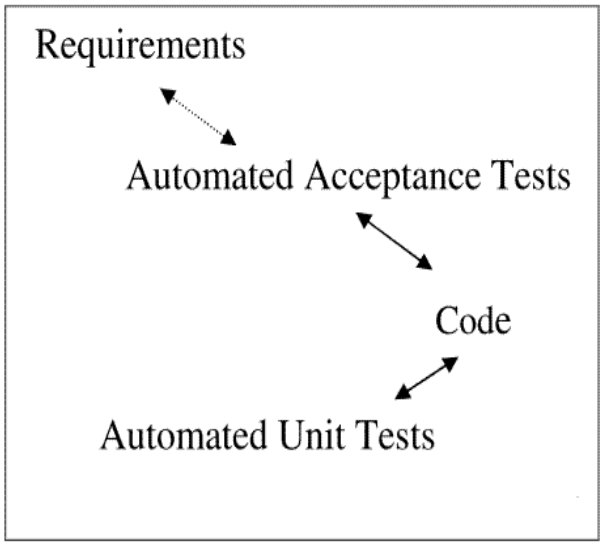

Figure 5 Traceability from requirements to code

Refactoring is an important aspect of TDD, but can represent a serious challenge to traceability. Refactoring of code may lead to the appearance of new traceability links and the disappearance of old traceability links between tests/requirements and code. Additionally, refactoring may lead to temporary code degradation, when some of the existing tests fail to pass. When refactoring, the TDD developer must ensure that all unit tests continue to pass, so unit tests might need to be refactored together with the source code.

IV. GUIDELINES FOR AGILE RE

This section introduces some guidelines to improve the performances of requirements engineering processes in agile environment and to enhance the quality of requirements.

- **Customer Involvement:** agile development focuses very strongly on customer interaction. At the beginning, all relevant ideal stakeholders have to be identified. Selecting the right customers and prioritizing their respective requirements is a key issue. The different elicitation practices aim to get as much knowledge as possible from all stakeholders and resolve inconsistencies.
- **Agile Projects Contracts:** at the beginning, the most critical requirements are expressed by the stakeholders as well as they can, so that the experienced project leaders can determine an initial cost for agile projects and guess the cost of later changes.
- **Frequent Releases:** frequently delivering parts of the system provides the ability to release faster expected results to the customers in order to get feedbacks from them. Hence, the requirements are implemented in an iterative and incremental fashion.
- **Requirements Elicitation Language:** use linguistic methods for requirements elicitation, derived from Natural Language Processing (NLP)

[7]. In other words, requirements are collected using the language of the customer, not a formal language for requirements specification.

- **Non-Functional Requirements (NFR):** in agile approaches handling of NFR is ill defined [9]. We propose the customers and agile team leaders to arrange for meetings to discuss NFR (and all critical requirements) in the earliest stages. Once the initial NFR of a project have been identified and documented, the agile teams can begin with development.
- **Smaller agile teams are flexible:** smaller agile teams allow continuous communications between them and stakeholders in efficient way, and the requirements changes are controlled. Fig. 6 shows that whenever the agile teams are smaller, the chances of the project success increased [16].

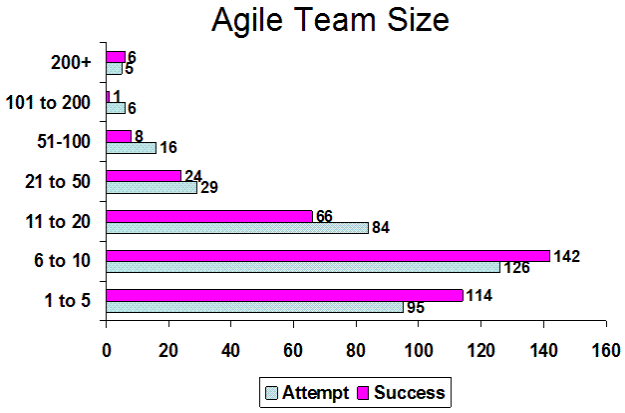

Figure 6 Agile team sizes

- **Evolutionary requirements:** RE in agile methods accommodate changing requirements even late in the development cycle, but that changes to the requirements must wait until the culmination of each iteration. Therefore, agile development does not spend much time in initial requirements elicitation. Consequently, this methodology will ensure that iterations are consistent with expectations, and that the development process will remain organized.
- **No early documentation:** any documents produced in the early stages can quickly become irrelevant because the agile principles encourage requirements change. By allocating only 5%-15% of the resources to requirements we think development team can still address shortcomings in agile development while complying with the agile principles in general.
- **Requirements splitting:** if the agile team considers a requirement too complex, this technique helps the customer to divide it into simpler ones. This helps agile teams to better understand the functionalities requested by the customer, and helps agile teams working in parallel with frequent communications between them. In XP [16], the requirements are written on story cards, the complex user stories are broken down smaller stories. Of course not all user stories can be divided since somecontain several sub-requirements, or record non-functional requirements. If a story card could be successfully divided, the original story card is discarded, since it no longer needed. All requirements are now included in the union of the new story cards.
- **Requirements Traceability:** a major upset in the development of large systems, especially those with evolving requirements is ensuring that the design of the system meets the current set of requirements. We are persuaded that agile projects would work better if they include requirements traceability tools together with validation tools. A good practice would be to identify the traceability links in TDD environment. In other words, the traceability links between test cases and related code should be identified and evolved to control co-changes. In this way, once the code is refactored, the agile team is able to re-build the traceability matrix again and determine what are the test cases needed to be re-run. In particular, the focus should be on the identification of the traceability links added or deleted after the refactoring process. In case the traceability links between source code and the related unit tests are broken during refactoring, this may be treated as a warning for possible code and/or unit test review [31]. Traceability information between requirements, source code and unit tests can also be used to drive software development, by identifying requirements for which unit tests and/or source code has not been implemented yet. In addition, traceability information can be used to support refactoring. Similar test cases can be grouped in test suite and traced onto source code classes. Source code classes related to more than one test suite are good candidates for refactoring.

V. CONCLUSION AND FUTURE WORK

The agile methodology manifesto supports a very efficient RE; this paper surveys the real process and activities of agile RE including feasibility study, elicitation, analysis, documentation, validation, and management. The secret of the success of agile RE is customer collaboration, good agile developers, and experienced project managers. This article provides some recommendations to solve the requirements documentati--on problem in agile projects, to make agile methodology suitable for handling projects with critical (functional and non-functional) requirements, to allow agile teams involved in large software projects to work in parallel with frequent communications between them. As future work, we will present industrial case studies that support our ideas, and try to develop a tool that support the distinction between functional and non-functional requirements; also we ignite debates for solving the traceability problem in TDD environment to re-establish

traceability after refactoring and to use traceability to improve refactoring.

AKNOWLEDGMENTS

We would like to thank Mr. Avishek Shrestha for his help, valuable ideas, and various references.

REFERENCES

[1] K. Beck, A. Cockburn, R. Jeffries, and J. Highsmith, (2001). "Agile manifesto". http://www.agilemanifesto.org, 23-02-2010.

[2] J. Highsmith, "Agile Software Development Ecosystems," *Addison–Wesley, Boston*, MA, 2002.

[3] J. Highsmith, K. Orr, A. Cockburn, "Extreme programming," *E-Business Application Delivery*, pp. 4–17, February, 2000.

[4] S. Bose, M. Kurhekar, J. Ghoshal, "Agile Methodology in Requirements Engineering," *SETLabs Briefings Online*. http://www.infosys.com/research/publica-tions/agile-requirements-engineering.pdf, February, 2010.

[5] A. Eberlein and J. Leite, "Agile Requirements Definition: A View from Requirements Engineering," Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02), Germany, 2002.

[6] R. Goetz, "How Agile Processes Can Help in Time-Constrained Requirements Engineering," *International Workshop on Time-Constrained Requirements Engineering*, 2002

[7] F. Paetsch, A. Eberlein, F. Maurer, "Requirements engineering and agile software development," *Eighth International Workshop on Enterprise Security, Linz, Austria*, 9 - 11 June, 2003.

[8] A. Sillitti, G. Succi, "Requirements Engineering for Agile methods," *Engineering and Managing Software Requirements, Springer*, 2005.

[9] B. Ramesh, L. Cao and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Info Systems J*, "doi:10.1111/j.1365-2575.2007.00259.x", 2007.

[10] B. Rompaey and S. Demeyer, "Establishing Traceability Links between Unit Test Cases and Units under Test," *13th European Conference on Software Maintenance and Reengineering, Germany*, 2009.

[11] S. Ambler, "When does(n't) agile modeling make sense?,".http://www.agilemodeling.com/essays/whenDoesAMWork.htm, February, 2010.

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

[13] S. R. Palmer and J. M. Felsing, "A Practical Guide to Feature-Driven Development," *The Coad Series*, 2002.

[14] Peter Coad, Eric Lefebvre and Jeff De Luca, "Java Modeling in Color with UML," *Prentice Hall PTR*, Chapter 6, 1999.

[15] Jennifer Stapleton, "DSDM - Dynamic System Development Method," *Addison-Wesley*, 1995.

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

[17] Pekka Abrahamsson, Outi Salo, Jussi Rankainen and Juhani Warsta, "Agile software development methods - Review and analysis," *VTT Electronics*, 2002.

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

[19] A. Polini, "Software Requirements," http://www1.isti.cnr.it/~polini/lucidiSE/Requirements1.pdf , February, 2010.

[20] A. Sidky and J. Arthur, "Determining the Applicability of Agile Practices to Mission and Life-Critical Systems," *Proceedings of the 31st IEEE Software Engineering Workshop, IEEE Computer Society*, pp. 3-12, 2007.

[21] T. Poppendieck and M. Poppendieck, "Lean Software Development: An Agile Toolkit for Software Development Managers," *Addison-Wesley*, 2003.

[22] O. Salo, "Systematical Validation of Learning in Agile Software Development Environment," *7th International Workshop on Learning Software Organizations, Kaiserslautern, Germany*, April, 2005.

[23] O. Salo and P. Abrahamsson, "Integrating Agile Software Development and Software Process Improvement: a Longitudinal Case Study," *4th International Symposium on Empirical Software Engineering, Noosa Heads, Australia*, November, 2005.

[24] I. Sommerville and P. Sawyer, "Requirements Engineering – A Good Practice Guide," *John Wiley & Sons*, 2000.

[25] E. Chikofsky and J. I. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17, 1990.

[26] K. Beck and M. Flower, "Planning Extreme Programming," *Addison-Wesley*, 2001.

[27] B. Boehm, "Verifying and validating software requirements and design specifications," *IEEE Software*, Vol. 1(1), 1984.

[28] L. Delgadillo, "Story-Wall: Lightweight Requirements Management for Agile Software Development," *15th IEEE International Requirements Engineering Conference*, pp.377-378, 2007.

[29] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 4, 2007.

[30] O. Gotel, and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proceedings of the IEEE International Conference on Requirements Engineering*, pp 94-101, April, 1994.

[31] J. Hayes, A. Dekhtyar and D. Janzen, "Towards Traceable Test Driven Development," *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pp.26-30, 2009.

[32] M. Fowler, "Refactoring: Improving the design of existing code," *Addison Wesley*, 1999.