An Evaluation Framework for Requirements Elicitation in Agile Methods

Waleed Helmy
Faculty of Computers & Information
Cairo University
Cairo, Egypt
w.helmy@fci-cu.edu.eg

Amr Kamel
Faculty of Computers & Information
Cairo University
Cairo, Egypt
a.kamel@fci-cu.edu.eg

Osman Hegazy
Faculty of Computers & Information
Cairo University
Cairo, Egypt
o.hegazy@fci-cu.edu.eg

***Abstract*—Gathering, understanding and managing requirements is a key factor to the success of a software development effort. There are several requirement techniques available for requirement gathering which can be used with agile development methods. These techniques concentrate on a continuous interaction with the customer to address the evolution of requirements, changing requirements, prioritizing requirements and delivering the most important functionalities first. However, problems have been reported with the use of the agile methods in the area of requirements elicitation particularly with an over reliance on a customer and lack of elicitation guidelines. This paper describe how requirements elicitation is usually done in more conventional software development processes and makes an evaluation framework for the way requirements elicitation can be done in the agile methods and this could result in improvements to agile approaches.**

Keywords-Agile Methods; Requirements Elicitation; Agile Requirements Elicitation.

I. INTRODUCTION

Agile software development approaches have become more popular during the last few years. Several methods have been developed with the aim to be able to deliver software faster and to ensure that the software meets customer changing needs. All these approaches share some common principles: Improved customer satisfaction, adapting to changing requirements, frequently delivering working software, and close collaboration of business people and developers [4, 9, 13].

Requirements engineering (RE), on the other hand, is a software engineering process with the goal to identify, analyze, document and validate requirements for the system to be developed [14]. Often, requirements engineering and agile approaches are seen being incompatible: RE is often heavily relying on documentation for knowledge sharing while agile methods are focusing on face-to-face collaboration between customers and developers to reach similar goals [1].

This paper aims to discuss how requirements elicitation techniques can be used within agile development context. Several studies addressed the requirements elicitation in agile methods. In [15], a new method for automatically retrieving functional requirements from the stakeholders using agile processes is presented. The presented method is a machine learning system for the automation of some aspects of the software requirements phase in the software engineering process. This learning system encompasses knowledge acquisition and belief revision in a knowledge base. The aim of the algorithm is to collect information from the various stakeholders and integrate a variety of learning methods in the knowledge acquisition process, while involving certain and plausible reasoning.

The goal oriented requirements engineering method proposed in [11] identifies the requirements in terms of goals which are well understood by the stakeholders and the goals are generally extracted from the stakeholders. While extracting the goals, the high level goals are decomposed/refined/broken to get the lower level goals/sub-goals involving active participation of stakeholders through the process of goal decomposition/refinement/splitting involving Agents.

Since micro-businesses have restrictions with their budget, manpower, and technical exposure to software, some trade-offs must be addressed. A novel approach in [16] demonstrated how several models and techniques such as goals, business process models, patterns, and non-functional requirements, have helped in defining the software requirements of the micro-business.

However, problems have been reported with the use of the agile methods in the area of requirements elicitation particularly with an over reliance on a customer and lack of elicitation guidelines. This paper describe how requirements elicitation is usually done in more conventional software development processes and makes an evaluation framework for the way requirements elicitation can be done in the agile methods and this could result in improvements to agile approaches.

The next section gives an overview on what the requirements elicitation is and what its techniques are. Section III states the agile manifesto. Section IV discusses agile methods from the requirements elicitation perspective. Section V summarizes the agile requirements engineering. In Section VI, we provide an evaluation framework for requirements elicitation in agile methods. Section VII summarizes the requirements elicitation issues in agile methods. The last section presents the conclusion.

II. REQUIREMENTS ELICITATION

Requirements engineering is concerned with identifying, modeling, communicating and documenting the requirements for a system, and the contexts in which the system will be used. Requirements describe what is to be

 ISBN: 978-1-61208-230-1

done but not how they are implemented [6]. There are many techniques available for use during the RE process to ensure that the requirements are complete, consistent and relevant. The fundamental principle underlying requirements engineering is that a system should be clearly specified before its design and implementation [11]. So, the aim of RE is to help to know what to build *before* system development starts in order to prevent costly rework. This goal is based on two major assumptions:

- The later mistakes are discovered the more expensive it will be to correct them [3].
- It is possible to determine a stable set of requirements before system design and implementation starts.

The RE process consists of five main activities [2]: Elicitation, Analysis and Negotiation, Documentation, Validation, and Management.

Requirements elicitation tries to discover requirements and identify system boundaries by consulting stakeholders (e.g., clients, developers, users). System boundaries define the context of the system. Understanding the application domain, business needs, system constraints, stakeholders and the problem itself is essential to gain an understanding of the system to be developed.

The most important techniques for requirements elicitation are described in the remainder of this section.

Interviews: Interviewing is a method for discovering facts and opinions held by potential users and other stakeholders of the system under development. Mistakes and misunderstandings can be identified and cleared up. There are two different kinds of interviews:

- The closed interview, where the requirements engineer has a pre-defined set of questions and is looking for answers
- The open interview, without any pre-defined questions the requirements engineer and stakeholders discuss in an open-ended way what they expect from a system.

In fact, there is no distinct boundary between both kinds of interviews. You start with some questions which are discussed and lead to new questions [8]. The advantage of interviews is that they help the developer to get a rich collection of information. Their disadvantage is that this amount of qualitative data can be hard to analyze and different stakeholders may provide conflicting information.

Observation and Social Analysis: Observational methods involve an investigator viewing users as they work and taking notes on the activity that takes place. Observation may be either direct with the investigator being present during the task, or indirect, where the task is viewed by some other means (e.g. recorded video). It is useful for studying currently executed tasks and processes. Observation allows the observer to view what users actually do in context. This overcomes issues with stakeholders describing idealized or oversimplified work processes.

Focus Groups: Focus groups are an informal technique where a small group of users from different backgrounds and with different skills discuss in a free form issues and concerns about features of a system prototype. Focus groups help to identify user needs and perceptions, what things are important to them and what they want from the system. They often bring out spontaneous reactions and ideas. Since there is often a major difference between what people says and what they do, observations should complement focus groups.

Focus groups can support the articulation of visions, design proposals and a product concept. Additionally, they help users in analyzing things that should be changed, and support the development of a 'shared meaning' of the system [7].

Brainstorming: Brainstorming helps to develop creative solutions for specific problems. Brainstorming contains two phases - the generation phase, where ideas are collected, and the evaluation phase, where the collected ideas are discussed. In the generation phase, ideas should not be criticized or evaluated. The ideas should be developed fast and be broad. Brainstorming leads to a better problem understanding and a feeling of common ownership of the result.

Prototyping: A prototype of a system is an initial version of the system which is available early in the development process. Prototypes of software systems are often used to help elicit and validate system requirements. There are two different types of prototypes. Throw-away prototypes help to understand difficult requirements. Evolutionary prototypes deliver a workable system to the customer and often become a part of the final system. Prototypes can be paper based (where a mock-up of the system is developed on paper), "Wizard of Oz" prototypes (where a person simulates the responses of the system in response to some user inputs) or automated prototypes (where a rapid development environment is used to develop an executable prototype).

III. AGILE MANIFESTO

The Agile manifesto as the Agile Manifesto official site states is as follows "*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

- ***Individuals and interactions*** *over processes and tools.*
- ***Working software*** *over comprehensive documentation.*
- ***Customer collaboration*** *over contract negotiation.*
- ***Responding to change*** *over following a plan.*

That is while there is value in the items on the right; we value the items in the left more." [9].

IV. AGILE METHODS

A. Extreme Programming

XP uses story cards for elicitation [1]. A user story is a description of a feature that provides business value to the

customer. Use cases, on the other hand, are a description of interactions of the system and its users and do not mandatory have to provide business value.

Before story cards can be written, customers have to think about what they expect the system to do. This process can be seen as brainstorming. Thinking about a specific functionality leads to more ideas and to more user stories. Every story is discussed in an open-ended way before implementation. Initially, developers ask for enough details to be able to estimate the effort for implementing the story. Based on these estimates and the time available, customers prioritize stories to be addressed in the next iteration. XP emphasizes writing tests before coding. Acceptance tests are defined by the customer and are used to validate the completion of a story card. XP is based on frequent small releases. This can be compared with requirements review and with evolutionary prototyping.

B. Scrum

The main Scrum techniques are the product backlog, sprints, and daily scrums [1]. With regard to Requirements Engineering the product backlog plays a special role in Scrum. All requirements regarded as necessary or useful for the product are listed in the product backlog. It contains a prioritized list of all features, functions, enhancements, and bugs. The product backlog can be compared with an incomplete and changing (a kind of "living") requirements document containing information needed for development. For each sprint (= 30 day development iteration), the highest priority tasks from the backlog are moved to the sprint backlog. No changes are allowed to the sprint backlog during the sprint. I.e. there is no flexibility in the requirements to be fulfilled during a sprint but there is absolute flexibility for the customer reprioritizing the requirements for the next sprint. At the end of a Sprint a sprint review meeting is held that demonstrates the new functionality to the customer and solicits feedback.

C. Feature Driven Development

Feature Driven Development (FDD) is a short iteration process for software development focusing on the design and building phase instead of covering the entire software development process [4]. 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 items of the feature list are prioritized by the team. The feature list is reviewed by domain members [5]. FDD proposes a weekly 30-minute meeting in which the status of the features is discussed and a report about the meeting is written. Reporting can roughly be compared with requirements tracking.

D. Agile Modeling

The basic idea of AM [2] is to give developers a guideline of how to build models that help to resolve design problems but not 'over-build' these models. Like XP, AM points out that changes are normal in software development. AM does not explicitly refer to any RE techniques but some of the practices support several RE techniques (e.g. tests and brainstorming). AM highlights the difference between informal models whose sole purpose is to support face-to-face communication and models that are preserved and maintained as part of the system documentation. The later are what is often found in RE approaches.

V. AGILE REQUIREMENTS ENGINEERING

The agile principles applied to software engineering include *iterative and incremental development, frequent releases of software, direct customer involvement, minimal documentation and welcome changing requirements even late in the development cycle [6]*.

Conventional RE processes focus on gathering all the requirements and preparing the requirements specification document up front before proceeding to the design phase. These up front requirements gathering and specification efforts leave no room to accommodate changing requirements late in the development cycle. On the other hand,

On the other hand, agile requirement engineering [12] aims at applying agile thoughts to traditional requirement engineering. It is the optimization and improvement of traditional requirement engineering, getting it fit to the continuous changes of requirements.

Agile RE welcomes changing requirements even late in the development cycle [3]. This is achieved by using the agile practice of *Evolutionary Requirements* which suggests that requirements evolve over the course of many iterations rather than being gathered and specified up front. Hence, changes to requirements even late in the development cycle can be accommodated easily.

Initially, the high level features for the system are defined where features indicate the expected functionality. All the features have to be identified upfront in order to determine the scope of the system. These features describe the expected functionality of business value to the customers. The development period spans multiple release cycles. Only one feature or a subset of the identified features is considered for development during a release cycle. Then, the requirements for each feature are gathered just-in-time (JIT) from the customers before the development of that feature. As only a subset of the identified features is implemented during a release cycle, only details of this subset of features are gathered from the customers. Customer are actively involved in the Agile RE process. Usually, a customer is available onsite to provide details of the features to the development team. Direct customer involvement facilitates the adoption of the JIT philosophy.

 ISBN: 978-1-61208-230-1

Agile RE accommodates rapidly changing requirements. Changes to requirements identified are logged and are implemented in the following iterations. As only a subset of features is implemented during a release cycle, changes to these features do not affect the other features that are yet to be built.

Agile RE focuses on minimal documentation. No formal Requirements Specification is produced. The features and the requirements are recorded on story boards and index cards. The artifacts produced depend on the project. Some Agile RE artifacts are paper prototypes, use case diagrams and data flow diagrams. However, if the client requires formal documentation to be produced, the development team strives to produce the same.

Verification and Validation (V&V) of requirements in agile RE is more of a validation process. The agreed standards used for verification are usually stated in the form of user stories and hence, V&V is more of a validation process. Validation is not explicit and is carried out just-in-time. There is no specification of explicit validation activities in agile RE. As the customer is usually available onsite, the features/ requirements can be validated as and when required. Some Agile RE practices are listed below:

Evolutionary Requirements – The requirements are allowed to evolve over time. All the requirements are not identified upfront. This practice is called No Big Requirements Up Front (BRUF).

Incremental and iterative implementation of requirements – Agile RE suggests incremental development of software. The development period is divided into release cycles and each release cycle spans multiple iterations. Hence, the requirements are implemented in an iterative and incremental fashion.

Accommodate change late in the development life cycle – The main objective of Agile RE is to accommodate changing requirements even late in the development cycle. Usually changes identified to features are logged and incorporated during the future iterations.

Minimal requirements documentation – Documentation is usually in the form of features or stories recorded on index cards. No formal requirements specification documents are produced. However, when employing third party organizations for performing maintenance activities, minimal documentation is a disadvantage.

Gather details just-in-time– The development team defers gathering details till the latest responsible moment. Only the details of the features to be implemented during a release cycle are gathered. Adopting JIT philosophy helps accommodate changing requirements.

Implicit Verification and Validation (V&V) – As mentioned earlier, V&V is more of a validation process. Validation is not carried out explicitly.

Treat requirements like prioritized stack – 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. These prioritized features are stored in a stack and ordered by their priorities.

Adopt user terminology – The features and requirements are recorded in the domain language of the user. This is done in order to help users understand the captured needs and requirements.

Direct customer involvement – Agile RE mandates the involvement of customers at every stage of the development process. As customers are involved throughout, the developers can gather details about the features just-in-time.

VI. AN EVALUATION FRAMEWORK

The table below is an evaluation framework for the requirements elicitation techniques in agile methods. The framework compares four agile methods with respect to requirements elicitation.

In the previous sections, we gave an overview on requirements elicitation techniques as well as on agile methods. Here, we now analyze potential synergies between these approaches.

TABLE 1: AN EVALUATION FRAMEWORK FOR REQUIREMENTS ELICITATION IN AGILE METHODS

Method / RE Practice	XP	Scrum	Agile Modeling	FDD
Req. Elicitation	*User Stories* Interview, Brainstorming	Product Backlog Interview	X Brainstorming	Feature List Interview

Customer involvement: The CHAOS report [7] showed the critical importance of customer involvement. Customer involvement was found to be the number one reason for project success, while the lack of user involvement was the main reason given for projects that ran into difficulties. A key point in all agile approaches is to have the customer 'accessible' or 'on-site'. Thus, traditional RE and agile methods agree on the importance of stakeholder involvement

Agile methods often assume an "ideal" customer representative: the representative can answer all developer questions correctly, she is empowered to make binding decisions and able to make the right decisions. Even if the requirements are elicited in group sessions (Scrum) it is not guaranteed that users or customers with all necessary backgrounds are present. On the other hand, RE has a less idealized picture of stakeholder involvement. The different elicitation techniques aim to get as much knowledge as possible from all stakeholders and resolve inconsistencies. In addition, RE uses externalization and reviews to ensure that all requirements are known and conflicting requirements are in the open

Another difference between traditional approaches and agile methods is that in traditional approaches the customer is mainly involved during the early phase of the project while agile methods involve the customer throughout the whole development process.

 ISBN: 978-1-61208-230-1

Feature: In agile development, a feature is a chunk of functionality that delivers business value. Features can include additions or changes to existing functionality. A feature should adhere to the following criteria:

- It should provide business value
- It should be estimable - it must have enough definition for the development team to provide an estimate of the work involved in implementing it
- It should be small enough to fit within an iteration - therefore, if it is too big, it should be broken down further
- It should be testable - you should understand what automated or manual test a feature should pass in order to be acceptable to the customer

The different agile methods use different terminology to refer to features. It is up to the team to decide which methodology or terminology to use. Extreme Programming uses the terms User Stories or Stories to represent features; Scrum uses Product Backlog to describe a feature list; Feature-Driven Development uses Feature. Ultimately, the goal is the same - to deliver business value regularly in small increments, and sooner rather than later.

Product Backlog, Features, User Stories: The product backlog is an ordered list of "requirements" that is maintained for a product. It contains product backlog Items that are ordered based on considerations like risk, business value, dependencies, date needed, etc. The features added to the backlog are commonly written in story format. The product backlog is the "What" that will be built, sorted in the relative order it should be built in. The product backlog contains rough estimates of business value and development effort, these values are often stated in story points.

Interviews As customer involvement is a primary goal of agile software development, the most common RE-related technique are interviews. Interviews provide direct and "unfiltered" access to the needed knowledge. It is known that chains of knowledge transfer lead to misunderstandings. All agile approaches emphasize that talking to the customer is the best way to get information needed for development and to avoid misunderstandings. If anything is not clear or only vaguely defined, team members should talk to the responsible person and avoid chains of knowledge transfer. Direct interaction also helps establishing trust relationships between customers and developers.

Brainstorming: This technique is not explicitly mentioned in any agile software development method but can be used with any approach.

VII. REQUIREMENTS ELICITATION ISSUES IN AGILE METHODS

The agile requirements elicitation approach toward requirements usually results in several architecture-related issues that can potentially have negative impact on architectural practices, artifacts or design decisions [10]. Following paragraphs describe the most commonly observed requirements elicitation issues when using agile approaches are:

Lack of focus on Non Functional Requirements: In agile approaches handling of non-functional requirements is ill defined [1]. Customers or users talking about what they want the system to do normally do not think about maintainability, portability, safety or performance. Some requirements concerning user interface or safety can be elicited during the development process and still be integrated. But most non-functional requirements should be known in development because they can affect the choice of database, programming language or operating system. Agile methods need to include more explicitly the handling of non-functional requirements in a way they can be analyzed before implementation.

Incomplete Requirements Elicitation: The "user stories" or the like are just the beginning points of both the requirements gathering and development processes in agile methods. Early requirements are simply a place to start. It is expected to add more requirements as more is known about the product. This attitude toward requirements makes software architecture development more difficult. The architecture that chosen by the team during the early cycles may become wrong, as later requirements becomes known [8].

VIII. CONCLUSION

This paper presented an evaluation framework of how the requirements are elicited in four common agile methods: XP, scrum, agile modeling, and FDD.

XP uses story cards for requirements elicitation through interviews and brainstorming techniques. In Scrum, all requirements regarded as necessary or useful for the product are listed in the product backlog. It contains a prioritized list of all features, functions, enhancements, and bugs. The requirements are elicited from users through interviews. In FDD, the overall domain model is developed that consists of class diagrams with classes, relationships, methods, and attributes. The methods express functionality and are the base for building a feature list. The feature list is elicited through interviews. AM does not explicitly refer to any RE techniques but some of the practices support several RE techniques (e.g. tests and brainstorming).

The agile requirements elicitation approach toward requirements usually results in several architecture-related issues that can potentially have negative impact on architectural practices, artifacts or design decisions [10]. The "user stories" or the like are just the beginning points of both the requirements gathering and development processes in agile methods. Early requirements are simply a place to start. It is expected to add more requirements as more is known about the product. Agile methods, however, have a lack of focus on certain parts of what is considered as important in requirements engineering. The customers don't usually cover non-functional requirements when they define requirements.

 ISBN: 978-1-61208-230-1

REFERENCES

[1] Eberlein, A., Maurer, F., and Paetsch, F., "Requirements Engineering and Agile Software Development", Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE CS Press, pp. 308-313, 2003.

[2] Scott, W., "Agile Modeling", John Wiley & Sons, 2001.

[3] Scott, W., "Agile Requirements Modeling", http://www.agilemodeling.com/essays/agileRequirements.htm , retrieved: October, 2012.

[4] Pekka, A., Outi, S., Jussi, R., and Juhani, W., "Agile software development methods - Review and analysis", VTT Publications, No. 478, 2002.

[5] Peter, C., Eric, L., and Jeff, L., "Java Modeling in Color with UML", Prentice Hall PTR, Chapter 6, 1999.

[6] Soundararajan, S." Agile Requirements Generation Model: A Soft-structured Approach to Agile Requirements Engineering". Master Thesis. Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, VA. 2008.

[7] Standish Group: Chaos Report, http:// www.standishgroup.com, retrieved: October, 2012.

[8] Tomayko, J., "Engineering of Unstable Requirements Using Agile Methods", International Conference on Time-Constrained Requirements Engineering, Essen, Germany, 2002.

[9] Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., "Manifesto for Agile software Development", http://www.agilemanifesto.org/, retrieved: October, 2012.

[10] Babar, M., "An exploratory study of architectural practices and challenges in using agile software development approaches", http://ulir.ul.ie/handle/10344/1127, retrieved: October 2012.

[11] Sen, M. and Hemachandran, K., "Elicitation of Goals in Requirements Engineering Using Agile Methods", Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops, pp. 263-268, 2010.

[12] Jun, L., Qiuzhen, W., and Lin, G., "Application of Agile Requirement Engineering in Modest-Sized Information Systems Development", Proceedings of the 2010 Second World Congress on Software Engineering - Volume 01 Pages 207-210, 2010.

[13] Poppendieck, T. and Poppendieck, M., "Lean software development: An agile toolkit for software development managers", Addison-Wesley, London UK, 2003.

[14] Kotonya, G. and Sommerville, I., "Requirements Engineering Processes and Techniques", John Wiley & Sons, Chichester, UK, 2002.

[15] Ankori, R., "Automatic Requirements Elicitation in Agile Processes", the IEEE International Conference on Software - Science, Technology & Engineering, pp. 101-109, 2005.

[16] Macaseat, R., Chung, L., Garrido, J., Noguera, M., and Luisa, M.," An agile requirements elicitation approach based on NFRs and business process models for micro-businesses, 12th International Conference on Product Focused Software Development and Process Improvement, pp. 50-56, 2011.

 ISBN: 978-1-61208-230-1