An Evaluation Framework for Requirements Envisioning 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*—A common agile practice is to perform some high-level requirements envisioning early in the project to gather and document business requirements during the initial phase of the project. The goals of requirements envisioning are to develop a common vision, identify the business goals, and identify the initial requirements for the system at a high-level. This paper presents an evaluation framework for the way the requirements envisioning can be done in the agile methods.**

Keywords-Requirements Envisioning; Agile Requirements Envisioning, Envisioning in Agile Methods.

I. INTRODUCTION

Information system development methodologies refer to a standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems [2]. They are developed to assure that software systems met established requirements. There are a number of methodologies used to develop and improve the systems such as the traditional waterfall, incremental development, prototyping, and spiral [13]. These methodologies impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. They do this by developing a detailed process with a strong emphasis on planning aspired by other engineering methodologies. However, these traditional systems development methodologies sometimes fall short in the new business environment [14]. They are too "heavy" to keep up with the pace of new business software development projects.

In response to the problems of the traditional methodologies, the agile methodology has evolved in the mid-1990s. Highsmith and Cockburn in [12] wrote that "*what are new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an agile world view*".

While Nerur and Balijepally in [3] defined agile methodologies as people-centric, that recognize the value of skilled people and their relationships bring to software development. They also explained in their paper that Agile methods focus on providing high customer satisfaction through three principles: quick delivery of quality software; active participation of concerned stakeholders; and creating and acting effectively toward changes. They added that big upfront designs, plans and extensive documentation are of little value to practitioners of agile methods. In [4] agile is defined as a software development method that is people-focused, communication-oriented, flexible (ready to adapt to expected change at any time), speedy (encourage rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive (reacts appropriately to expected and unexpected changes), and learning (focuses on improvement during and after product development).

However, this paper focuses only on the agile methods and the way the requirements envisioning can be done in the agile methods. A common agile practice is to perform some high-level requirements envisioning early in the project to help come to a common understanding as to the scope of what you're trying to accomplish. The goals at this point are to identify the business goals for the effort, develop a common vision, and quickly identify the initial requirements for the system at a high-level. This initial requirements envisioning effort is on the order of hours or days, not weeks or months, as we see on traditional projects.

The next section presents the agile software development life cycle. Section III gives an overview on agile requirements envisioning. Section IV discusses four agile methods namely XP, Scrum, Feature Driven Development, and Agile modeling. Section V presents an evaluation framework for requirements envisioning in four agile methods. The last section presents the research conclusion.

II. AGILE SDLC

Agile software development life cycle is comprised of six phases: Iteration -1/Pre-Planning, Iteration 0/Warm Up, Construction, Release/End Game, Production, and Retirement [11]. Here is a description of each phase.

A. Iteration -1: Pre-Project Planning

This phase includes the following activities:

- Define the business opportunities
- Identify a viable for the project
- Assess the visibility

 ISBN: 978-1-61208-230-1

B. Iteration 0/Warm Up: Project Initiation

The first week or so of an agile project is often referred to as “Iteration 0” (or "Cycle 0"). The goal during this period is to initiate the project by:

- Garnering initial support and funding for the project.
- Starting to build the team.
- Setting up the environment.

C. Estimating the project: Construction Iterations

During construction iterations agilists incrementally deliver high-quality working software which meets the changing needs of our stakeholders. This can be achieved by:

- Collaborating closely with both our stakeholders and with other developers
- Implementing functionality in priority order
- Analyzing and designing
- Ensuring quality
- Regularly delivering working software
- Testing, testing, and yes, testing.

D. Release Iterations(s): The "End Game"

During the release iteration(s), also known as the "end game", we move the system into production. There are several important aspects to this effort:

- Final testing of the system.
- Rework. There is no value testing the system if you don't plan to act on the defects that you find. You may not address all defects, but you should expect to fix some of them.
- Finalization of any system and user documentation.
- Training. We train end users, operations staff, and support staff to work effectively with our system
- Deploy the system

E. Production

The goal of the Production Phase is to keep systems useful and productive after they have been deployed to the user community. This process will differ from organization to organization and perhaps even from system to system, but the fundamental goal remains the same: keep the system running and help users to use it.

F. Retirement

The goal of the Retirement Phase is the removal of a system release from production, and occasionally even the complete system itself, an activity also known as system decommissioning. Retirement of systems is a serious issue faced by many organizations today as legacy systems are removed and replaced by new systems. You must strive to complete this effort with minimal impact to business operations. If you have tried this in the past, you know how complex it can be to execute successfully.

III. AGILE REQUIREMENTS ENVISIONING

Agile requirements activities are evolutionary (iterative and incremental) and highly collaborative in nature. Initially, requirements are explored at a high level via requirements envisioning at the beginning of the project and the details are explored on a just-in-time (JIT) basis via iteration modeling and model storming activities. The strategy is to take advantage of modeling, which is to communicate and think things through without taking on the risks associated with detailed specifications written early in the lifecycle, a traditional practice referred to as "Big Requirements Up Front" [1]. For the first release of a system you need to take several days, with a maximum of two weeks for the vast majority of business systems, for initial requirements and architecture envisioning. There are several models to envision the requirements which are:

High-level use cases (or user stories) - The most detail that we would capture would be point form notes for some of the more complex use cases, but the majority just might have a name. The details are best captured on a just-in-time (JIT) basis during construction.

User interface flow diagram - This provides an overview of screens and reports and how they're inter-related.

User interface sketches - Sketch out a few of the critical screens and reports to give your stakeholders a good gut feeling that you understand what they need.

Domain model - A high-level domain model shows major business entities and the relationships between them. Listing responsibilities, both data attributes and behaviors, can be left until later iterations.

Process diagrams - A high-level process diagram that shows some of the critical processes, are likely needed to understand the business flow.

Use-case diagram - Instead of a high-level process diagram you might want to do a high-level use case diagram instead.

IV. AGILE METHODS

Agile methods, generally, promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices that allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals [9].

A. Extreme Programming

XP uses story cards for elicitation [5]. 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.

 ISBN: 978-1-61208-230-1

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 [5]. 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 potentially shippable product is delivered and a sprint review meeting is held that demonstrates the new functionality to the customer and solicits feedback [10]

C. Feature Driven Development

As the name implies, features are an important aspect of Feature Driven Development (FDD). A feature is a small, client-valued function. Features are to FDD as use cases are to the Rational Unified Process (RUP) and user stories are to XP – they're a primary source of requirements and the primary input into your planning efforts.

FDD is a short iteration process for software development focusing on the design and building phase instead of covering the entire software development process [6]. 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 [7]. 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 Agile Modeling (AM) is to give developers a guideline of how to build models that help to resolve design problems but not 'over-build' these models [8]. 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. AN EVALUATION FRAMEWORK

Table 1 presents an evaluation framework for requirements envisioning in agile methods. The framework compares four agile methods: Scrum, extreme Programming, Feature Driven Development, and Agile Modeling Driven Development with respect to three evaluation criteria: Activities, Participants, and Time Frame.

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

Criteria / Agile Method	Activities	Participants	Time Frame
XP	Initial Requirements Modeling, Initial Architecture Modeling	All Team Members	Hours or days
Scrum	X	All Team Members	Hours or days
FDD	Build an Object Model	All Team Members	Hours Or days
AMDD	Initial Requirements Envisioning, Initial Architecture Envisioning	All Team Members	Hours Or days

XP encompasses the initial requirements modeling and initial architectural modeling aspects of the agile software development lifecycle. This phase includes development of the architectural spike and the development of the initial user stories. From a requirements point of view it suggests that you require enough material in the user stories to make a first good release and the developers should be sufficiently confident that they can't estimate any better without actually

 ISBN: 978-1-61208-230-1

implementing the system. Every project has a scope, something that is typically based on a collection of initial requirements for your system. Although the XP lifecycle does not explicitly include a specific scope definition task it implies one with user stories being an input into release planning. User stories are a primary driver of the XP methodology – they provide high-level requirements for your system and are the critical input into your planning process. The implication is that you need a collection of user stories, anywhere from a handful to several dozen, to get your XP project started. The second aspect of the exploration phase focuses on your system architecture. The architecture within an XP project is less formal than in traditional methodologies, with a preference for keeping your system flexible – XP recommends that you embrace change, whereas architecture-driven approaches advise you to build the skeleton the system first because some things are difficult to change. The XP approach is to identify a metaphor that describes how you intend to build your system. The metaphor acts as a conceptual framework, identifying key objects and providing insight into their interfaces. The metaphor is defined during an architectural spike early in the project, during the first iteration or during a pre-iteration that is sometimes referred to as a zero-feature release (ZFR).

In scrum project life cycle, there are no structured activities for requirements envisioning. The requirements are treated like a prioritized stack, pulling just enough work off the stack for the current iteration. At the end of the iteration, the system is demoed to the stakeholders to verify that the work that the team promised to do at the beginning of the iteration was in fact accomplished. But, where does the product backlog come from? It is actually the result of initial requirements envisioning early in the project.

An FDD project starts by building overall domain model to envision the requirements. The goal of envisioning is to identify the scope of the effort, the initial architecture, and the initial high-level plan. As with other agile software development processes, systems are delivered incrementally by FDD teams.

The envisioning activity in agile modeling includes two main sub-activities, initial requirements envisioning and initial architecture envisioning [4]. These are done during iteration 0, iteration being another term for cycle or sprint. The envisioning effort is typically performed during the first week of a project, the goal of which is to identify the scope of your system and a likely architecture for addressing it. To do this you will do both high-level requirements modeling and high-level architecture modeling. The goal isn't to write detailed specifications that prove incredibly risky in practice, but instead to explore the requirements and come to an overall strategy for your project. Finally and as shown in table 1, we need to take several days, with a maximum of two weeks for the vast majority of business systems, for initial requirements and architecture envisioning.

VI. CONCLUSION

Agile requirements envisioning aims to develop a high level understanding of the project. This allows the initial identifications of requirements for the system at the beginning of the project. This paper presented an evaluation framework for requirements envisioning in four agile methods: XP, Scrum, Feature Driven Development, and Agile Modeling. The results showed that clear specification of activities in the agile requirements envisioning process is missing and there is a lack of a set of activities and techniques that practitioners can choose from. Hence, there is a need to develop a structured approach that clearly outlines the activities of the agile requirements envisioning process and suggests techniques or practices that can be used.

REFERENCES

[1] Ambler, S., "Examining the "Big Requirements Up Front (BRUF) Approach", http://www.agilemodeling.com/essays/examiningBRUF.htm, retrieved: October, 2012.

[2] Hoffer, H., George, J., and Vlacich, J., "Modern Systems Analysis and Design", Pearson Prentice Hall, 2006.

[3] Nerur, S. and Balijepally, V., "Theoretical reflections on agile development methodologies", Communication of the ACM, vol. 50, pp. 79-83, 2007.

[4] Qumer, A. and Sellers, B., "An evaluation of the degree of agility in six agile methods and its implacability for method engineering", Information and Software Technology, 2007.

[5] 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, 2003.

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

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

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

[9] http://en.wikipedia.org/wiki/Agile_software_development, retrieved: October, 2012.

[10] Meng, X.,Wang, Y., Shi, L., and Wang, F., "A process pattern Language for Agile Methods", 14 th Asia-Pacific Software Engineering Conference, 2007.

[11] Ambler, S., "The Agile System Development Life Cycle", www.ambysoft.com/essays/agileLifecycle.html, retrieved: October, 2012.

[12] Highsmith, J. and Cockburn, A., "Agile Software Development: The Business of Innovation", IEEE Computer, Vol. 34, No. 9, pp. 120 – 127, 2001.

[13] Sommerville, I., "Software Engineering", Addison Wesley, 7th edition, 2004.

[14] Seyam, M. and Galal-Edeen, G., "Traditional versus Agile: The Tragile Framework for Information Systems development", the International Journal of Software Engineering (IJSE), Vol. 4, No. 1, pp. 63-93, ISSN: 1687-6954, 2011.

 ISBN: 978-1-61208-230-1