Requirement Gathering for small Projects using Agile Methods

Kavitha C.R
Dept of Computer Applications
SNGIST
N Parur

Sunitha Mary Thomas
Dept of Computer Applications
Christ Knowledge City
Airapuram

ABSTRACT
Gathering, understanding and managing requirements is a key factor to the success of a software development effort. Requirement engineering is a critical task in all development methods including the agile development method.

There are several requirement techniques available for requirement gathering which can be used with agile development methods. These techniques concentrates on a continuous interaction with the customer to address the evolution of requirements, changing requirements, prioritizing requirements and delivers the most important functionalities first.

This article presents an overview of agile software development methods and a best requirement elicitation technique used for requirement capturing. We present an application case of requirement gathering process by using User stories for web-based, cost-effective and efficient software (ISODTA- ISO documentation teaching automation) which automates the ISO documentation of teaching process at the institution SNGIST using SCRUM, an agile software development methodology.

Keywords
Agile methodologies, Scrum, requirement elicitation, user stories, story index card

1. INTRODUCTION
Agile development methodology is an approach used in software development which has become more popular during the last few years. Agile software development is iterative and incremental development where you do development in small iterations. It attempts to provide many opportunities to assess the different stages of a project throughout the software development life cycle. These methods aim to deliver software faster and ensure that the software meets customer's changing needs and expectations. Agile methodologies focus on skills, communication and community clarifying the roles of customers, managers and developers for more satisfying and productive relationship.

Agile Methods

The agile methodologies are also known as the lightweight methodologies. These methodologies consist of development techniques designed to deliver products on time, on budget, and with high quality and customer satisfaction. There are different agile methodologies available today. The most popular flavours include:

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Feature Driven Development (FDD)
- Scrum

The Agile Manifesto

The Agile Manifesto is a statement of the principles that underpin agile software development. It was drafted from 11 to 13 February 2001, at The Lodge at the Snowbird ski resort in the Wasatch Range of mountains in Utah, where representatives of various new methodologies (such as Extreme Programming, Scrum, DSDM, Adaptive Software Development, Crystal, Feature Driven Development, and Pragmatic programming) met to discuss the need for lighter alternatives to the traditional heavyweight methodologies.

"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 on the left more".

Agile Methods (AMs) Vs Traditional Methods (TMs).

TABLE 1. AMs VS TMs

Traditional Methodologies(TMs)	**Agile Methodologies (AMs)**
Focused on processes, sequence of processes and useful tools	Focused on people
Flexible only in the beginning of project	Emphasizes the communication, collaboration, rapid exchange of information, team work and the functioning software and flexibility
Help deliver projects on time and budget but not suitable when you are moving to a new technology or for changing requirements	Uncertain budgets and unclear milestones
Use single model (like waterfall model)	Involves a lot of different methods that work in a similar way.

2. AGILE SOFTWARE DEVELOPMENT AND REQUIREMENT HANDLING

2.1 Introduction

Agile development works in a different manner. Instead of specifying everything before you start developing, one need to take a small portion of the most important features and only specify and implement those. Requirement gathering is different for different methods, for. E.g. in SCRUM & XP, user stories are used. Requirement specification is an iterative process that continues until the customer is satisfied with the product. By using this method, it is easier to react to changes and can have a better estimate and better control on the development leading to a better quality product and customer satisfaction.

Since Agile development is a fairly new technique there is no good answers on how to handle requirements and researches are being undertaken in this area.

2.2 Extreme Programming

Extreme Programming abbreviated as XP, was invented by Kent Beck which addresses the specific needs of software development conducted by small teams in the face of vague or changing requirements

2.2.1 Advantages of XP:

- it stresses customer satisfaction
- empowers your developers to confidently respond to changing customer requirements, even late in the life cycle
- emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team
- solve the problem efficiently
- follow simple rules

XP improves a software project in five essential ways; communication, simplicity, feedback, respect, and courage. Extreme Programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested.

2.2.2 Core Practices

There are twelve core practices that define Extreme Programming. The practices can be described as a cycle of activities. The inner circle describes the tight cycle of the Programmers. The outer loop describes the planning cycle that occurs between the Customers and Programmers. The middle loop shows practices that help the team communicate and coordinate the delivery of quality software.

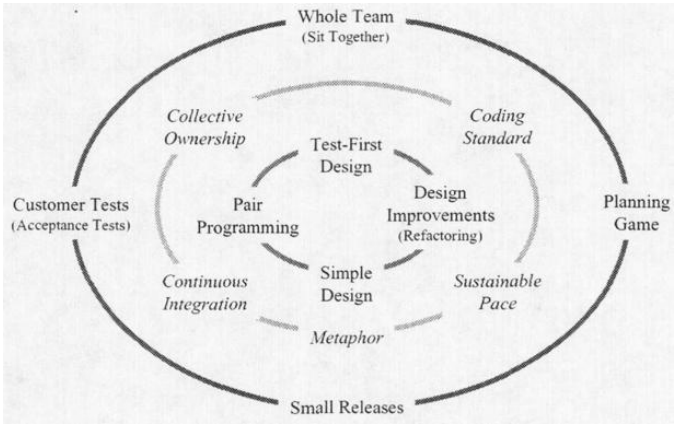

Figure 1. XP Practices and the Circle of Life from [8]

2.3 Adaptive Software Development (ASD)

Adaptive Software Development (ASD) was developed in 2000 by Jim Highsmith which has grown out of the Rapid Application Development (RAD). Like other agile methodologies, ASD aims to increase a software organization's responsiveness while decreasing development overhead. It embodies the belief that continuous adaptation of the process to the work at hand is the normal state of affairs.

ASD replaces the traditional waterfall cycle with a repeating series of speculate, collaborate, and learn cycles. This dynamic cycle provides for continuous learning and adaptation to the emergent state of the project. The characteristics of an ASD life

cycle are that it is mission focused, feature based, iterative, time boxed, risk driven, and change tolerant.

The word "**speculate**" refers to the paradox of planning – it is more likely to assume that all stakeholders are comparably wrong for certain aspects of the project's mission, while trying to define it. **Collaboration** refers to the efforts for balancing the work based on predictable parts of the environment (planning and guiding them) and adapting to the uncertain surrounding mix of changes caused by various factors – technology, requirements, stakeholders, software vendors, etc. The **learning** cycles, challenging all stakeholders, are based on the short iterations with design, build and testing. During these iterations the knowledge is gathered by making small mistakes based on false assumptions and correcting those mistakes, thus leading to greater experience and eventually mastery in the problem domain.

2.4 Dynamic Systems Development Method (DSDM)

2.4.1 Introduction

Dynamic Systems Development Method (DSDM) developed in the United Kingdom in the 1990s by the DSDM Consortium, an association of vendors and experts in the field of software engineering created with the objective of "jointly developing and promoting an independent RAD framework" by combining their best practice experiences. It is a software development methodology originally based upon the Rapid Application Development methodology. DSDM is an iterative and incremental approach that emphasizes continuous user involvement.

Its goal is to deliver software systems on time and on budget while adjusting for changing requirements along the development process.

2.4.2 Phases of DSDM

The DSDM framework consists of three sequential phases:

Phase 1 - The Pre-Project phase where candidate projects are identified, project funding is realized and project commitment is ensured.

Phase 2 - The Project life-cycle phase include five stages to create an information system.

Phase 3 - Post-project phase ensures the system operating effectively and efficiently.

2.4.3 Four stages of the Project life-cycle

2.4.3.1 Stage1A: The Feasibility Study, where the feasibility of the project for the use of DSDM is examined.

Stage1B: The Business Study, which examines the influenced business processes, user groups involved and their respective needs and wishes.

2.4.3.2 Stage2: Functional Model Iteration, where the requirements that have been identified in the previous stages are converted to a functional model.

2.4.3.3 *Stage3: Design and Build Iteration*, integrates he functional components from the previous phase into one system that satisfies user needs.

2.4.3.4 Stage4: Implementation, where the tested system including user documentation is delivered to the users and training of future users is realized.

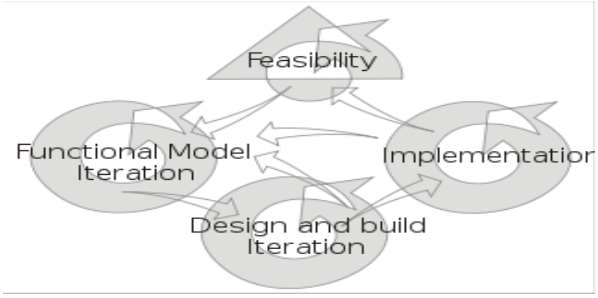

Figure 2. Project Life Cycle from [9]

2.5 Feature Driven Development (FDD)

2.5.1 Introduction

Feature Driven Development (FDD) was initially developed by Jeff De Luca, which is an iterative and incremental software development process. FDD blends a number of industry-recognized best practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner.

2.5.2 Best practices

Feature-Driven Development is built around a core set of industry-recognized best practices, derived from software engineering. These practices are all driven from a client-valued feature perspective. It is the combination of these practices and techniques that makes FDD so compelling. The best practices that make up FDD are shortly described below.

- **Domain Object Modeling**, explores and explains the domain of the problem to be solved. The resulting domain object model provides an overall framework in which to add features.

- **Developing by Feature**. Any function that is too complex to be implemented within two weeks is further decomposed into smaller functions until each sub-problem is small enough to

be called a feature. This makes it easier to deliver correct functions and to extend or modify the system.

- **Individual Class (Code) Ownership**. It means that distinct pieces or grouping of code are assigned to a single owner who is responsible for the consistency, performance, and conceptual integrity of the class.

- **Feature Teams**. It is a small, dynamically formed team that develops a small activity. By doing so, multiple minds are always applied to each design decision and also multiple design options are always evaluated before one is chosen.

- **Inspections**. Inspections are carried out to ensure good quality design and code, primarily by detection of defects.

- **Configuration Management**. It helps with identifying the source code for all features that have been completed to date and to maintain a history of changes to classes as feature teams enhance them.

- **Regular Builds**. It ensures there is always an up to date system that can be demonstrated to the client and helps highlighting integration errors of source code for the features early.

- **Visibility of progress and results**. By frequent, appropriate, and accurate progress reporting at all levels inside and outside the project, based on completed work, managers are helped at steering a project well.

2.6 Scrum

2.6.1 Introduction

Scrum is an agile approach to software development. It has been found out that Scrum is very beneficial when applied to small and medium projects. Rather than a full process or methodology, it is a framework which instead of providing complete, detailed descriptions of how everything is to be done on the project, much is left up to the team, because the team will know best how to solve its problem. For e.g., in a sprint planning meeting is described in terms of the desired outcome (a commitment to set of features to be developed in the next sprint. Scrum relies on a self-organizing, cross-functional team. There is no team leader in a scrum team who decides which person will do which task or how a problem will be solved.

2.6.2 Unique about Scrum

Of all the agile methodologies, Scrum is unique because it introduced the idea of "empirical process control." That is, Scrum uses the real-world progress of a project — not a best guess or uninformed forecast — to plan and schedule releases. In Scrum, projects are divided into succinct work cadences, known as sprints, which are typically one week, two weeks, or three weeks in duration. At the end of each sprint, stakeholders and team members meet to assess the progress of a project and plan its next steps. This allows a project's direction to be adjusted or reoriented based on completed work, not speculation or predictions.

The Scrum methodology really works is due to a set of roles, responsibilities, and meetings that never change. If Scrum's capacity for adaption and flexibility makes it an appealing option, the stability of its practices give teams something to lean on when development gets chaotic.

2.6.3 The Roles of Scrum

Scrum has three fundamental roles: Product Owner, ScrumMaster, and team member.

- **Product Owner:** In Scrum, the Product Owner is responsible for communicating the vision of the product to the development team. He or she must also represent the customer's interests through requirements and prioritization. Because the Product Owner has the most authority of the three roles and also has the greatest responsibility. In other words, the Product Owner is the single individual who must face the trouble when a project goes awry and also must answer questions from the team.

- **ScrumMaster:** The ScrumMaster acts as a liaison between the Product Owner and the team. The ScrumMaster does not manage the team. Instead, he or she works to remove any impediments that are obstructing the team from achieving its sprint goals. Scrummaster helps the team to remain creative and productive and makes sure of its successes visible to the Product Owner. The ScrumMaster also works to advise the Product Owner about how to maximize return over investment for the team.

- **Team Member:** In the Scrum methodology, the team is responsible for completing work. Ideally, teams consist of seven cross-functional members, plus or minus two individuals. For software projects, a typical team includes a mix of software engineers, architects, programmers, analysts, Quality Assurance experts, testers, and User Interface designers. Each sprint, the team is responsible for determining how it will accomplish the work to be completed. This grants teams a great deal of autonomy, but, similar to the Product Owner's situation, that freedom is accompanied by a responsibility to meet the goals of the sprint.

Scrum projects make progress in a series of sprints, which are time boxed iterations no more than a month long. At the start of a sprint, team members commit to delivering some number of features that were listed on the project's product backlog. At the end of the sprint, these features are done--they are coded, tested, and integrated into the evolving product or system. At the end of the sprint a sprint review is conducted during which the team demonstrates the new functionality to the product owner and other interested stakeholders who provide feedback that could influence the next sprint.

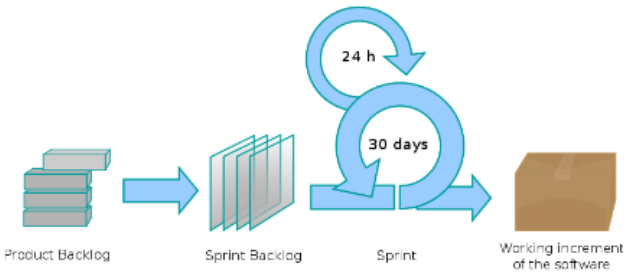

Figure 3. Scrum Process from [10]

In agile software development methodology, there will be a "potentially shippable product increment" at the end of each iteration/sprint. During each iteration, the Scrum Master (SM) should ensure that the work the team committed to do during that sprint is what the team actually delivers.

- The SM should ensure that the team understands the user stories and provides accurate estimates.
- Once the team commits to a unit of work (and is in a sprint), the SM should make sure that the team is focused only on the user stories that are within the current sprint.
- During the sprint, the SM should also ensure that the team is focused on the sprint work (and only the sprint work). He/she should eliminate any impediments that might be blocking the team's progress.
- The SM should maintain the task burndown chart and story burnup charts, along with the team's velocity calculations.

2.6.4 Requirement Gathering in SCRUM

In Scrum, user stories can be used to capture a project's ever changing requirements. Because project needs fluctuate, Scrum doesn't consider requirements gathering to be a one-time task. Instead, Scrum focuses on the incremental unfolding of requirements. The work is decomposed and captured at different levels in varying levels of detail during the project. At each level, the work is broken down into smaller, more manageable pieces of work.

Product backlog (PB): High-level capabilities and features are captured as user stories. The PB is the repository of all the work that needs to happen during the project.

Sprint backlog: The team identifies the user stories that will be worked on during the current iteration, or sprint. Each user story is decomposed further into tasks.

3. USER STORIES
3.1 Introduction

User stories are used in agile software development methodologies like XP and Scrum. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it. It describes all those functionalities that are valuable to a stakeholder. It is a reminder to have a conversation with the customer. In short, user stories are very slim and high-level requirements artefacts. It shifts the focus from writing to talking. It supports and encourages iterative development.

The basic components of a User Story includes

- Cards (physical medium)
- Conversation (discussion surrounding stories)
- Confirmation (tests and validation that verify story completion).

User stories are often written on 3''X 5'' index cards which are very easy to work with. Simple index cards can be used to write the user stories and a small index card box can be used to store these cards arranged by priority. One has to try to keep the user stories short and to the point. During the sprint planning sessions, cards can be re-prioritized if needed.

Project Name:	Story Card ID:
Priority:	Estimation:

Front side of the index card

Acceptance Test:
1. …… 2. …… 3. ……..

Back Side of the index card

Figure: 4, Story Index Card

User stories should be validation-centric. The team needs to identify and write down the validation conditions for each user story. The story should not be closed until these validation conditions are met satisfactorily.

3.2 INVEST Model

A well-written user story follows the INVEST model. . A user story should be:

Independent: A good story should stand on its own.

Negotiable: A good story is negotiable. It is not an explicit contract for features; rather, details are co-created by the customer and programmer during development. A good story captures the essence, not the details.

Valuable: A user story should be valuable to the business.

Estimable: The team should be able to gauge the size of a story.

Small: A good rule of thumb is that a user story should not be more than eight to sixteen hours of effort.

Testable: The story should include validation criteria.

Well-written User Stories are cornerstones for Agile Development. They should be independent of each other; the details should be negotiated between the users and the developers; the stories should be of value to the users; they should be clear enough for developers to be able to estimate them; they should be small; and they should be testable through the use of pre-defined test cases.When writing individual, detailed tasks under each user story, teams should focus on writing **SMART** tasks.

Specific
Measurable
Achievable
Relevant
Time-boxed

4. OUR CASE STUDY

4.1 Introduction

ISO 9000 provides a framework that can be used by any size or type of organization to develop a quality system. Today, ISO 9000 standards are rapidly being implemented in many service industries such as educational institutions in particular. Benefits from implementing ISO 9000 standards can be divided into four different parts as benefits to system, faculty, students and external benefits to the organizations. Today, customers expect quality in all aspects of life. The customer wants to be assured that educational institutions provide quality service. This is an era of competition and globalization of knowledge much importance is given to the quality. As a result educational institutions have started implementing ISO.

Quality and accountability in higher education are inevitably going to be the principal themes in the higher education policy. The objective of ISO documentation is mainly to provide the service continuously as before even though the Institution decides to replace personnel all together. The control of documentation is the critical element to retain the ISO registration. So, good customized software is required to do the documentation process efficiently and effectively.

In this project of requirements gathering, since the customers are present onsite, user stories has been used for requirement gathering on story cards. Like in XP, user stories can be applied in SCRUM to capture requirements. SCRUM treats the requirements like a prioritized task. It freezes the requirements for the current iteration to provide a level of stability for the developers. In Scrum, work is expressed in the backlog as user stories. User stories document requirements with particular attention to the end user's point of view. By using user stories, one would be able to focus on exactly what the user need/want without going into details on how this should be done. A team may write its user stories in a number of ways as long as they are written from the perspective of the end user.

User stories generally follow the following template:

As a … (role or actor) (**Who**)

I want … (what capability or feature do they need) (**What**)

so that … (why is it of business value or benefit) (**Why**)

"As a *type of user* I want *capability or feature* so that *business value or benefit*"

The "Who" and "What" are essential to the story, but the "Why" only helps with clarity and sets up the acceptance test.

4.2 Applying User Stories in the Case Study

Stories help you ask the right questions about the context and reason for the request from the prospective of the person requesting the feature.

Here are some examples of User Stories for the web based ISODTA software:

• As a new faculty to SNGIST, I want to register myself to get a username and password so that I can use ISODTA.

• As a faculty, I want to select the names of the students from the database so that I can make them as a batch of some particular class which I am going to handle.

• As a faculty, I want to mark the attendance of the students of my class so that I can calculate the monthly attendance and monthly attendance percentage

• As a faculty, I want to enter the marks of series exam, model exam, assignments, presentations, project and seminar of the students so that I can calculate the internal marks of the students.

• As a faculty, I want to prepare the lesson plan so that the course can be completed within the stipulated time.

• As a faculty, I want to record the shortfall topics so that I can take the corrective action.

• As a faculty, I want to generate reports on the absentees so that I can take suitable actions.

• As a faculty, I want to record the details of the assignment, seminars, case study, group discussion etc.

• As a faculty, I want to conduct the result analysis of internal examination so that I can find out the performance of the class.

• As a faculty, I want to analyze the University exam result so that I can get an idea about the results of my subject for a few years.

• As a faculty, I want to mark the subject hours in the time table.

• As a batch coordinator, I want to prepare the time table for my class.

• As a batch coordinator, I want to prepare the test plan so that all the exams can be planned and conducted within schedule.

- As a batch coordinator, I want to prepare the weekly report so that shortfalls of different subjects can be found out
- As a batch coordinator, I want to prepare the monthly attendance statement so that attendance shortage of students can be traced for each subject.
- As a batch coordinator, I want to prepare the series exam and model exam marks statement.

4.3 Acceptance Test/Criteria for User Stories

As a vital part of the planning our project, user stories define what we are going to build in our project. User stories are prioritized to indicate which are most important for the system and are broken down into software engineering tasks and estimated by the development team. When we implement a user story a more formal acceptance test will be written to ensure that the goals of the story are fulfilled. Acceptance tests are created from user stories.

Acceptance criteria are specified indicators or measures employed in assessing the ability of a component, structure or system to perform its intended function. They are the expectations of the product owner on what will be delivered. Acceptance criteria can include functionality that the system will perform, interface look and feel and necessary documentation.

These are high-level tests to test the completeness of a user story or stories 'played' during any sprint/iteration. These tests are created ideally through collaboration between business customers, business analysts, testers and developers; however the business customers (product owners) are the primary owners of these tests. As the user stories pass their acceptance criteria, the business owners can be sure of the fact that the developers are progressing in the right direction about how the application was envisaged to work and so it's essential that these tests include both business logic tests as well as User Interface validation elements (if need be).

Acceptance test cards are ideally created during sprint planning or iteration planning meeting, before development begins so that the developers have a clear idea of what to develop.

During iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests. A story can have one or many acceptance tests, whatever it takes to ensure the functionality works 100%. Each acceptance test represents some expected result from the system. We must verify the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority. A user story is not considered complete until it has passed its acceptance tests.

5. CONCLUSION

Developing software that meets the customers or stakeholders' needs and expectation is the ultimate goal of the software development methodology. To meet their need we have to perform requirement engineering which helps to identify and structure requirements. In many cases it is risky or very difficult and not economical to produce a complete, verifiable set of requirements. Traditional software development has a problem to deal with requirement change after careful analysis and negotiation.

Generally customers have rarely a general picture of the requirements or system in their mind which leads problems related to requirements like requirements conflicts, missing requirements, and ambiguous requirements etc, and does not address non-functional requirements from exploration phase. This problem is well tackled by SCRUM as SCRUM recommends an on-site customer to represents their requirements through user stories on story cards. Scrum places a high value on customer interaction and satisfaction. User stories, which are concise often, dwelling within the boundaries of the index card, are considered the best method for explaining requirements when using the scrum approach.

6. REFERENCES

[1] Pressman, R. S. (2005), Software Engineering: A Practitioner's Approach, Sixth Edition, McGraw-Hill International Edition.

[2] Alford M. W, A requirements engineering methodology for realtime process requirement, IEEE transactions on software engineering volume 3

[3] Cohn, M (2009), Succeeding with Agile: Software Development using Scrum

[4] Cohn, M (2003) User stories applied for Agile Software Development 2003 Addison-Wesley

[5] The agile manifesto http://WWW.agilemanifesto.org/ (cited 2010-07-21)

[6] Extreme Programming- a gentle introduction http://WWW.extremeprogramming.or/ (cited 2010-08-03)

[7] Kishore S, Naik R, Software Requirements and Estimation

[8] http://gannman.x10hosting.com/Portfolio/ExtremeProg.pdf (cited 2010-09-11)

[9] http://en.wikipedia.org/wiki/DSDM (cited 2010-09-23)

[10] http://en.wikipedia.org/wiki/Scrum_(development) (cited 2010-09-24)

[11] http://en.wikipedia.org/wiki/INVEST_(mnemonic) (cited 2010-10-12)

[13] Scrum (development) http://en.wikipedia.org/wiki/Scrum_%28development%29 (cited 2010-09-11).