# Requirements Engineering Methodology in Agile Environment

**Waleed Helmy, Amr Kamel and Osman Hegazy**

**Faculty of Computers and Information, Cairo University**
**Giza, Postal Code: 12613, Egypt**

**Faculty of Computers and Information, Cairo University**
**Giza, Postal Code: 12613, Egypt**

**Faculty of Computers and Information, Cairo University**
**Giza, Postal Code: 12613, Egypt**

**Abstract**

This paper provides a better understanding of the architecture-related issues in agile projects and proposes a methodology to guide and assist practitioners adopting agile requirements engineering. The methodology was motivated by the lack of structure to the agile requirements engineering process with minimal impact on agility. It describes in detail the phases in the agile requirements engineering process and suggests techniques that can be used to perform these phases. As the length of the development lifecycle is taken into account, the methodology describes not only the requirements engineering process activities but the complete development process as well. It reflects the agile principles such as direct stakeholder involvement, evolutionary requirements, refactoring, no BRUF, just-in-time gathering of details and minimal documentation.

***Keywords***: *Architecture Challenges in Agile Environment, Agile Requirements Engineering*

## 1. Introduction

The conventional approach to the RE process focuses 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 consume long time and leave no room to accommodate changing requirements later in the development cycle. Some of the issues faced by organizations involved in up front requirements gathering and specification efforts are [8]:

- Requirements change over a period of time due to changes in customer and user needs, technological advancement and schedule constraints. Identifying new requirements or changing existing requirements affects the other requirements as new dependencies may surface. Also, changes to requirements involves modifying the architecture and in turn, the code. Accommodating changing requirements is an expensive activity. Hence, gathering all the requirements up front is expensive in the face of rapidly changing requirements. Also, new requirements can be identified late in the development cycle. Requirements Management activities help plan for and control change. However, it is not always possible to avoid changes to requirements.

- Rapidly changing business environments can cause requirements to become obsolete before project completion.

On the other hand, the agile requirements engineering [8] welcomes changing requirements even late in the development cycle. 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. Agile requirements engineering has the following issues:

- Clear specification of activities in the agile requirements engineering process is missing and there is a lack of a set of techniques that practitioners can choose from.
- This approach toward requirements usually results in several architecture-related issues that can potentially have negative impact on architectural practices, artifacts or design decisions [8].

## 2. Requirements Issues in Agile Methods

Little is known about how agile projects conduct requirements engineering [1]. Recent studies have identified several problems that could result from the lack of detailed requirements specifications [2]. The following paragraphs describe the requirements issues when using agile approaches.

- **Missing Requirements Engineering Activities:** clear specification of activities in the agile requirements engineering process is missing and there is a lack of a set of techniques that practitioners can choose from.
- **Missing Requirements Interface:** agile methods assume that it is very hard to elicit all requirements from the user upfront. They also assume that such requirements evolve over time as the customer may change its mind [6]. So, nobody knows the entire requirements at the beginning of the project. That leads to missing the interface between requirements. As a consequence, the impact on the next iterations may cause re-work as the requirements interfaces were not addressed.
- **Non-Functional Requirements Elicitation**: agile methods do not provide any widely accepted technique for eliciting and managing non-functional requirements [3]. After every iteration, the product is released and the customer is able to test the product. If he identifies problems related to non-functional qualities, the team can adapt the system to meet such requirements in the subsequent iteration without affecting too much the schedule. Often, the customer does not perceive as high impact many non-functional requirements (e.g., scalability, security, etc.). This may affect deeply the release of the final version of the application. This approach to non-functional requirements may represent

a major risk for agile methods, since they lack of specific techniques for their management.

## 3. Architecture Challenges in Agile Methods

The agile requirements engineering approach toward requirements usually results in several architecture-related issues that can potentially have negative impact on architectural practices, artifacts or design decisions [7]. Following paragraphs briefly describe the most commonly observed architecture-related difficulties when using agile approaches.

- **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 [4].
- **Incorrect Prioritization of User Stories**: one of the key architecture-related challenges commonly experienced by agile teams is that User Stories may be prioritized without taking the technical considerations into account [7]. If critical interdependencies among User Stories are discovered later on, it usually requires significant re-factoring with consequences for the whole structure of the software.
- **Lack of Focus on Non Functional Requirements**: as mentioned, handling of non-functional **requirements** in agile approaches is ill defined [5]. 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.

## 4. Motivation towards the Proposed Model

The methodology was motivated by the lack of structure to the agile requirements engineering process with minimal impact on agility. It describes in detail the phases in the agile requirements engineering process and suggests techniques that can be used to perform these phases. As the length of the development lifecycle is taken into account, the methodology describes not only the requirements engineering process activities but the complete development process as well. The methodology reflects the agile principles such as direct stakeholder involvement, evolutionary requirements, refactoring, no BRUF, just-in-time gathering of details and minimal documentation.

As shown in figure 1, the methodology consists of eight phases. The first six phases (Inception, Feature List Identification, Feature Grouping, Group Prioritization, NFRs Identification, and Architecture Envisioning) cover the requirements and architecture envisioning part while the two remaining phases (Task Identification and Task Development) cover the requirements development part.

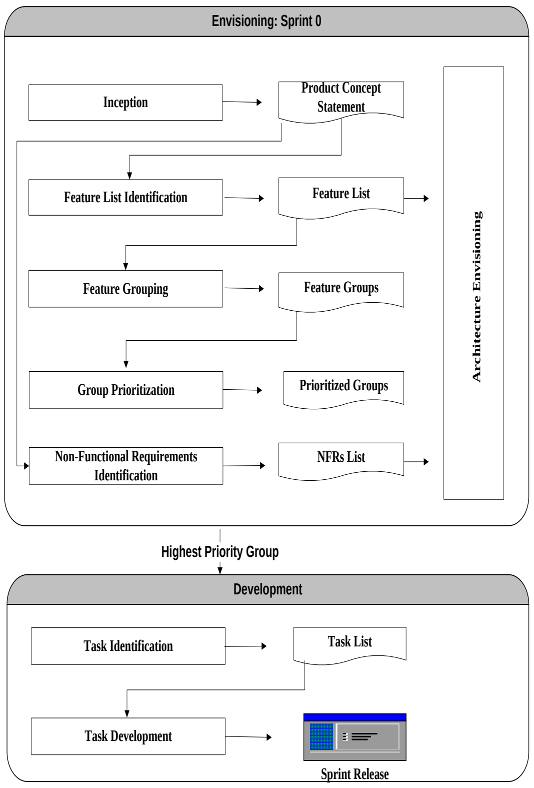

Fig. 1 The Proposed Methodology Structure

## 5. The Proposed Methodology

Figure 2 shows the proposed methodology complete life cycle. The main characteristic of the proposed methodology is that it provides a complete product development cycle. Each phase of the proposed methodology is shown in the figure. Sprint zero is used before the start of the development process. It is basically designed to envision system requirements and architecture. The requirements are presented in different levels which are: product level, feature level, story level, and task level. Table 1 summarizes all levels with the description linked to each.

The proposed methodology provides complete steps for sprint zero. Sprint zero starts with the inception phase and ends with a group or subset of it to be developed. The inception phase defines vision and goals for the project and a high level product mission statement is created which serves as the input to the feature list identification phase. Each item in the feature list includes definition of customer requirements that allows development team to produce a reasonable estimate of the effort during the development. The features identified during the feature list identification phase are classified into groups during the feature grouping phase. The groups are then prioritized and only one group is chosen for development during a release. As well, the architecture is envisioned in sprint 0 based on the feature list and the non-functional requirements identified in feature list identification phase and non-functional requirements identification phase respectively.

Each feature in the selected group is decomposed into stories. Each story then is decomposed into tasks. Tasks outline the details required for implementing the stories. These tasks are then developed during the task development phase using test driven development approach. The customers and developers perform acceptance testing to ensure that the system meets the customer and user needs.

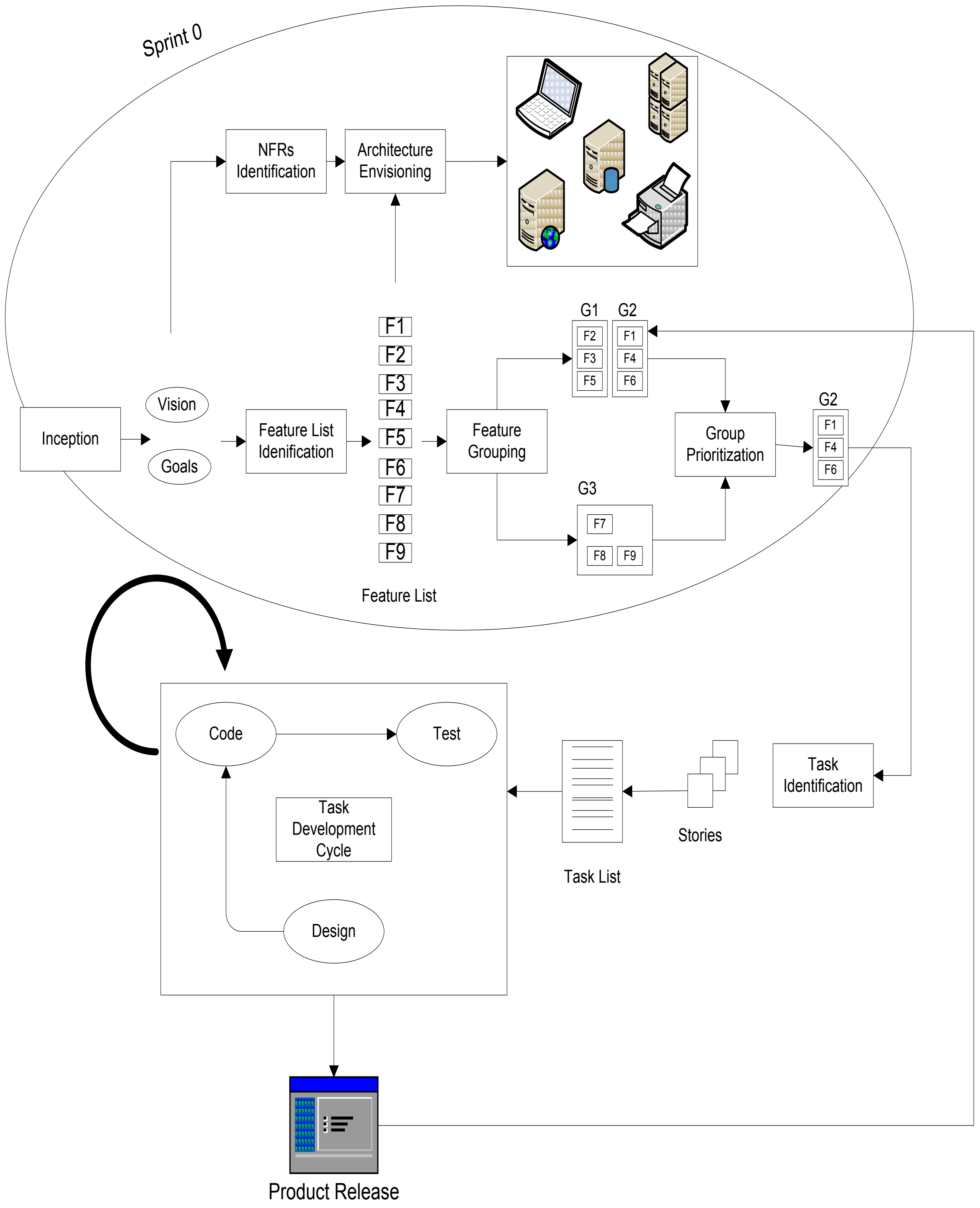

Fig. 2 The Proposed Methodology Life Cycle

Table 1: Requirements Levels

| Level | Name | Description |
|---|---|---|
| 1 | Product Level | Vision and Goals that form requirements |
| 2 | Feature Level | Features that the product support, but not too detailed or set of functionalities that provide business value to the customer |
| 3 | Story Level | Brief descriptions of customer valued functionality |
| 4 | Task Level | The details required for implementing the story |

The next sections describe in details the proposed methodology basic phases, activities, and techniques.

## 5.1 Inception Phase

The inception phase is the first phase of the proposed methodology and is designed to help the stakeholders establish relationship. This phase is carried out up front before starting the project. It is essentially a meeting or a series of meetings where all the stakeholders participate. During this phase, the customer is informed about the proposed methodology main phases, activities, and techniques. Goals for the project are formed and a vision statement is created that defines the entire project. The project team is assembled and the team members are empowered by having roles and responsibilities assigned to them.
As shown in figure 3, the basic activities of the inception phase are:

- **Assign Roles and Define Responsibilities –** Roles are assigned to team members and responsibilities are defined.
- **System Users Identification** – Identify the different users who will the use the system. Different **users** of the system have different needs. Hence, it is important to identify the various users of the system in order to identify their needs.
- **Proposed Methodology Explanation** – The customers and users need to be familiarized with the proposed methodology. This allows the customers and users to understand the methodology steps as they are involved through the complete software development process.

- **Establish Product Concept Statement** – The product concept statement is identified by system users. It contains the product mission statement and the goals for the project.

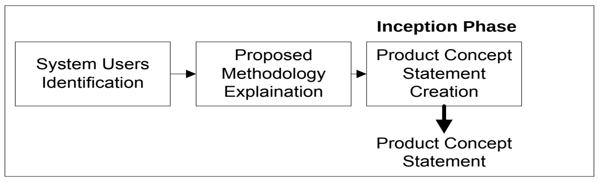
Inception Phase
System Users Identification
Proposed Methodology Explaination
Product Concept Statement Creation
Product Concept Statement

Fig. 3 Inception Phase

The output of this phase is a high level product mission statement which is a brief descriptive summary of the product. It would answer the following questions:

- Who are the product users?
- What will the product do?
- What problem(s) will the product solve?

The high level mission statement serves as the input to the features list identification phase. The stakeholders determine the expected functionality of the system from this statement.

## 5.2 Feature List Identification

The feature list identification is the second phase of the proposed methodology where the system features are identified. A feature can be defined as the smallest set of functionality that provides business value to the customer. The high level product mission statement created during the inception phase serves as the input to the feature list identification phase. The stakeholders identify the expected functionality from the high level product mission statement.
Features should be identified upfront in order to be informed about the scope of the project and plan for the release cycles. Identifying features upfront gives the “big picture” about the project. The features usually are identified over a series of meetings. The identified features are validated, the time for their completion estimated and are prioritized based on their value to the customer and stored in a prioritized feature list stack. Features are ordered in the stack based on their priorities. The Interviews, focus groups, and brainstorming are effective techniques to elicit business features from users.
As shown in figure 4, the various activities of the feature list identification are:

- **Preparation** – This activity is to arrange for a meeting among all the stakeholders to identify the major features of the system to be built.
- **Elicitation** – Release level features are identified using various techniques used for requirements elicitation. *Brainstorming* sessions involving all the stakeholders are generally an effective way to identify features. *Open-ended interviews* and *focus groups* with the customers and users are other techniques to elicit features. This process takes into account the agile practice of no BRUF and the details are gathered on a just-in-time basis.
- **Validation and Estimation** – The elicited features are validated and the time required to complete each feature is estimated. The validation process involves the development team discussing the identified features with the customers and users. The time estimates for a feature factor in time for gathering details from customers, coding, testing and helping customers plan and automate acceptance tests.
- **Prioritization** – Customers prioritize the identified set of features based on their needs and return on investment. Those features that have highest market value are given the top priority. Prioritization of features should take into account strong dependencies among feature sets.

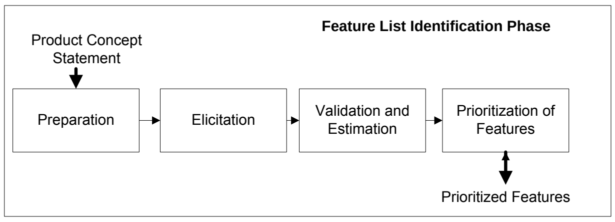
Feature List Identification Phase
Product Concept Statement
Preparation
Elicitation
Validation and Estimation
Prioritization of Features
Prioritized Features

Fig. 4 Feature List Identification Phase

The output of the feature list identification is a prioritized feature list which contains the prioritized features ordered by their priorities. The output of this phase will be the input to the feature grouping phase.

## 5.3 Feature Grouping

The identified features in the feature list identification phase are collected into groups. The groups are set of related features that serve a business area. The groups are validated and the time of their completion is initially estimated. Only

one group or a subset of the identified groups is chosen for development during a release.
As shown in figure 5, the various activities of this phase are:

- **Preparation** – This objective of this activity is to arrange for a meeting among the stakeholders is to create groups for the prioritized features.
- **Grouping** – The prioritized features are grouped into a set of groups. Groups are created for all the user features identified during the feature list identification phase.
- **Validation and Estimation** – The groups created are validated and the team estimates the time required for completing each group. Validation involves discussing the groups with the customers.

The output of this phase is feature groups that serve as the input to the group prioritization phase.

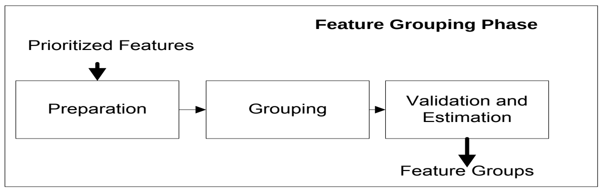

Fig. 5 Feature Grouping Phase

## 5.4 Group Prioritization

The team determines the dependencies if any among the groups identified in the feature grouping phase and prioritize the groups accordingly. The team also considers the customer and user preferences. The team and customers may have different sequences in which they would like to implement the groups. If there is a conflict, the customer is given preference. The customer is made aware of issues that may arise when their choices are given precedence. Only one group or a subset of the prioritized group is chosen for development during iteration. The output of this phase is prioritized stack list. As shown in figure 6, the various activities of this phase are:

- **Preparation** – This objective of this activity is to arrange for a meeting among the stakeholders is to prioritize the identified feature groups.
- **Prioritization** – The feature groups are prioritized and only one group or a subset of the prioritized group is chosen for development during iteration

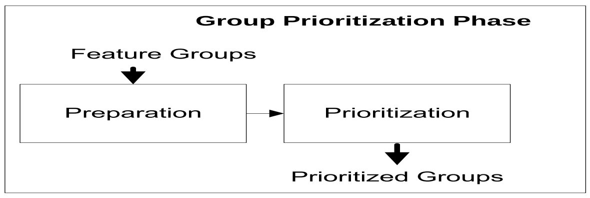

Fig. 6 Group Prioritization Phase

## 5.5 Non-Functional Requirements Identification

The phase aims to identify the system non functional requirements (NFRs). These NFRs are set of constraints on the software. The high level product mission statement created during the inception phase serves as the input to the non functional requirements identification phase. The stakeholders with the team identify the NFRs from the high level product mission statement. The identified NFRs are validated and prioritized based on their value to the customer and stored in a prioritized NFRs list stack. The Interviews and brainstorming are effective techniques to elicit NFRs from users.
As shown in figure 7, the various activities of this phase are:

- **Elicitation** – NFRs are identified using various **techniques** used for requirements elicitation. *Brainstorming* sessions involving all the stakeholders are generally an effective way to identify features. *Open-ended interviews* with the customers and users are other techniques to elicit NFRs.
- **Validation**– The elicited NFRs are validated. The validation process involves the development team discussing the identified NFRs with the customers and users.
- **Prioritization** – Customers prioritize the identified set of NFRs based on their needs.

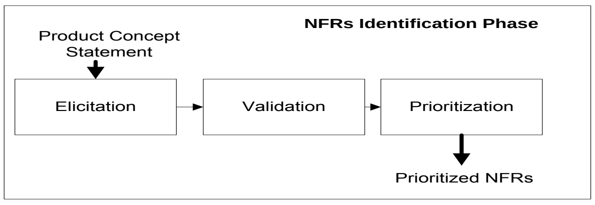

Fig. 7 NFRs Identification Phase

The output of the non functional requirements identification is a prioritized NFRs list which contains the prioritized NFRs ordered by their priorities. The output of this phase will be the input to the architecture envisioning phase.

## 5.6 Architecture Envisioning

The list of features identified in the feature list identification phase, the user stories identified in the task development phase, and the list of non-functional requirements identified in the non-functional requirements identification phase serve as the input to the architecture envisioning.
The goal of the architecture envisioning phase is to try to identify an architecture that has a good chance of working. This enables to set a feasible technical direction for the project and to provide sufficient information to organize the team around the architecture. The envisioned architecture presents the system technical infrastructure and the major business entities and their relationships to explore potential architecture-level requirements
Unlike agile methods in which the initial requirements and the initial architect models need to be evolved as you learn more about the project, the architecture envisioning phase of the proposed methodology tries to stabilize the architecture at the beginning of the project. As a result, this will reduce the risk of updating the system architecture every time a new requirement appears.
The **technology stack diagram** or **deployment diagram** are useful when doing initial architectural modeling because they depict the major software and hardware components and how they interact at a high level.
Part of the initial architectural modeling efforts, particularly for a business application, will include the development of high-level **domain model**. This model should capture the main business entities and the relationships between them. The initial domain model will be used to help guide both the physical data model as well as the class design
The architecture envisioning seems to provide a comprehensive view of the system being developed, while maintaining the agility of the development process rather than the heavy design documents that were always recommended in the traditional methodologies. To conclude this phase, below is a summary of benefits of applying the architecture envisioning phase:

- Ensure **better** understanding of the problem and the proposed solution
- Give an overall view of the issue of the information system being built, rather than the narrow vision of just coding the required software

- Increase the communication and knowledge transfer **among** the whole team

## 5.7 Task Identification

The prioritized groups identified in the group prioritization phase serve as the input to the task identification phase. Only one group or a subset of it is chosen for development during iteration. Each feature in the selected group is decomposed into stories. Stories are descriptions of user- or customer-valued functionality. Stories are defined at a lower level of abstraction when compared to the features. If multiple teams are involved, then each team can work on identifying stories for a particular feature. Each story is a sentence or two or a paragraph at most. Stories are recorded on index cards.

Each story then is decomposed into tasks by the development team. The task list for each story is essentially a to-do list created for the developers. Though the stories are themselves small, they are further disaggregated into tasks due to the following reasons:

- Each story maybe developed by more than one developer due to time constraints or developer skill sets. Hence, there is a need to further decompose stories into tasks.
- Decomposing stories into tasks ensure that the developers do not overlook any detail

The task lists provide details about the functionality to be implemented to the developers to guide them during the development of the tasks. Task lists for more than one story can be created in parallel by multiple teams involved in the development process. This enables faster software development. The development team brainstorms to create the task lists for each story. The team reviews the lists to ensure that they have not overlooked any details. These details are gathered just-in-time.

As shown in figure 8, the various activities of this phase are:

- **Preparation** – This objective of this activity is to arrange for a meeting among the stakeholders is to create stories and tasks for a a group of prioritized features.
- **Stories Identification** – The chosen feature is decomposed into stories. Techniques for eliciting stories include *interviews, observation, questionnaires and story-writing workshops*. A story writing workshop is a brainstorming session involving all the stakeholders dedicated to creating stories. Customers write the acceptance criteria for each story. Hence, the customers write the acceptance criteria and later validate the developed code against these criteria.

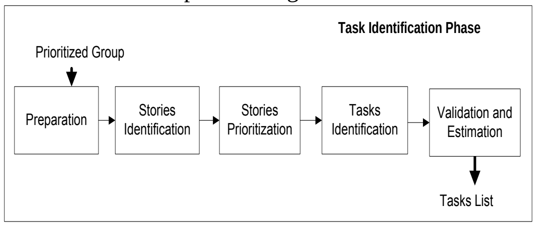

Fig 8 Task Identification Phase

- **Stories Prioritization** – The developers determine the dependencies if any among the stories and prioritize the stories accordingly. They also consider the customer and user preferences. Developers and customers may have different sequences in which they would like to implement the stories. If there is a conflict, the customer is given preference. The customer is made aware of issues that may arise when their choices are given precedence. The output of this activity is a **prioritized story stack**.
- **Tasks Identification** – The story is decomposed into tasks. Tasks outline the details required for implementing the stories.
- **Validation and Estimation** – The tasks created are validated and the developers estimate the time required for completing each task. Validation involves discussing the tasks with the customers, creating screen designs and paper prototypes.

## 5.8 Task Development

The tasks created during the tasks identification phase are developed in this phase. TDD is an agile practice and is a widely used approach to writing code. Also, as delivering product of value to the customer is a fundamental agile principle, Customer Acceptance Testing is of great importance. The proposed methodology reflects the agile philosophy and hence, we suggest using TDD and Customer Acceptance Testing as activities during this phase.

The developers follow TDD to implement the tasks. The customers and developers then test the available system against the acceptance criteria created previously. Each developer chooses a set of tasks for the story to be implemented based on their skills.

TDD is a combination of Test First development and refactoring [21]. Test First Development is a development technique where developers create a unit test first for a story or task before writing code. Refactoring is an agile practice which deals with changing the design or structure of the code without changing its result. Refactoring involves rewriting the code to improve its structure, while explicitly preserving its behavior. It improves the understandability of the code or changes its internal structure and design, and removes dead code, to make it easier for human maintenance in the future. Using TDD, developers, create tests first, then write code and then refactor the code in order to improve its structure. After refactoring, errors if any in the code are corrected. The developers do not write code before a test fails.

The following are the objectives of the task development phase:

For a story i and its task j:

- **Create tests** – The developers create unit tests first before writing code.
- **Write code** – The developers write **code** to satisfy the unit tests created.
- **Perform customer acceptance tests**

The activities carried out in the task development phase are:

- **Test Driven Development** (TDD) – “Clean code that works” is the goal of TDD.
- **Customer Acceptance Tests** – **Acceptance** tests ensure that the system developed meets the expectations of the customer. The customers create acceptance criteria for the stories and test the stories against the criteria.

## 6. Methodology Time Framework

For short projects (less than 6 months in length) you may do cycle zero in few hours and for medium and long projects (six months or more) you may decide to invest few days in this effort depending on the project size.

## 7. Proposed Methodology – Agile Manifesto

This section discusses how the proposed methodology is reflective of the values and principles stated in the Agile

Manifesto. The following paragraphs discuss the focal values and principles of the Agile Manifesto and how the proposed methodology reflects the agile philosophy.

- **"Individuals and Interactions over Processes and Tools"** and **"Customer Collaboration over Contract Negotiation"** are two of the focal values stated in the manifesto. The agile movement focuses on close team relationships, close working environments and direct stakeholder involvement. Human aspects of the software development process are considered more important than the process itself. Relationships between the customers and the development team are given preference over strict contracts. Rather than focusing on strict contracts, agile methods emphasize on collaborating with the customers to discuss the functionality expected of the system. These values are reflected in the proposed methodology in the following ways:
  - The methodology emphasizes direct customer involvement. Customers and users are involved throughout the process.
  - The activities described in the inception, features identification, features grouping, groups prioritization, and non functional requirements identification are essentially meetings among stakeholders to discuss, elicit and validate the customer and user needs.
  - Face-to-face communication among the members of the development team is encouraged especially during the features list identification.
- **"Working Software over Comprehensive Documentation"** is another value stated in the manifesto. The main objective of the software development team is to produce working software at regular intervals. Agile methods focus on iterative and incremental development. Working software is used to measure progress and minimal documentation is produced. The proposed methodology emphasizes on delivering working software to the customers at the end of each iteration. Documentation produced during the phases is minimal and mostly consists of product concept statement, feature list, groups and their prioritization, NFRs, high level architecture, and task list.

- "**Responding to change over following a Plan**": The methodology accepts changes in the task list by adding, removing, or updating data requirements or changing business logic or business steps. Also it encourages customer to change the feature list or feature stories if these changes have no impact on the architecture envisioned in sprint zero.

## 8. Proposed Methodology – Agile Development Practices

Agile practices such as incremental development and test-driven development are adopted for implementing system features. The proposed methodology adopts these practices which are common to the agile approach to software development. The agile development practices adopted by the proposed methodology are discussed below:

- **Test-Driven Development** – Developers write tests first **before** writing code. The proposed methodology suggests using TDD in the Development Phase to implement features.
- **Customer Acceptance Tests** – Acceptance tests ensure that the system developed meets the **expectations** of the customer. During the task development, the customers specify acceptance criteria for each feature. The customers later test the working software produced at the end of each iteration against the criteria specified previously.
- **Incremental Development** – Agile methods emphasize on incremental development and the proposed methodology accommodates the same. The development period consists of release cycles and working software is produced at the end of each release cycle. Incremental development is practiced as features group identified during the features grouping activity.
- **Gather Details Just-In-Time** – In the proposed methodology, the development team postpones gathering feature details till the task identification phase.
- **Direct Customer Involvement** – The proposed methodology encourages the **involvement** of customers at every stage of the development process. As customers are involved throughout, the team can gather details about the features just-in-time.
- **Evolutionary Requirements** – in the proposed methodology, 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).
- **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.

## 9. Conclusion

The conventional software development models try to define all essential requirements in the beginning of the software development phase by using huge effort. Agile methods, on the other hand, does requirements engineering in iterations: the requirements are defined in detail only when they are implemented.
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. Non-functional requirements are not precisely handled in agile methods and it would be good to concentrate more on these.

## References

[1] J. Erickson, K. Lyytinen, and K. Siau, “Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research” J. *Database Management*, 2005, vol. 16, no. 4, pp. 88–99.
[2] J. Nawrocki et al., “Extreme Programming Modified: Embrace Requirements Engineering Practices,” *Proc. IEEE Joint Int’l Conf. Requirements Eng.* (RE 02), 2002, IEEE CS Press, pp. 303–310.
[3] A. Eberlein, F. Maurer, F. Paetsch, , "Requirements Engineering and Agile Software Development", Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003
[4] J. Tomayko, “Engineering of Unstable Requirements Using Agile Methods”, *International Conference on Time-Constrained Requirements Engineering*, (2002) ,Essen, Germany, 9 September.

[5] A. Eberlein, F. Maurer, F. Paetsch, "Requirements Engineering and Agile Software Development", Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003

[6] K. Beck, "*Extreme Programming Explained: Embrace Change"*, Addison- Wesley, 1999

[7] M. Babar,"Going Agile? Beware of architecture-related changes and challenges", (2009b). Available at: http://www.acubecommunity.org/wikis/index.php/Architecturecentric_Methods_ and_Agile_Approaches.

[8] S.Ambler, "Agile Requirements Modeling", 2012 available at: http://www.agilemodeling.com/essays/agileRequirements.htm.