Agile Requirements Generation Model: A Soft-structured Approach to Agile Requirements Engineering

Shvetha Soundararajan

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Science and Applications

July 31, 2008
Blacksburg, VA

Keywords: Requirements Engineering, Agile Philosophy, Agile Methods,
Agile Requirements Engineering

Agile Requirements Generation Model: A soft-structured approach to Agile Requirements Engineering

Shvetha Soundararajan

ABSTRACT

The agile principles applied to software engineering include *iterative and incremental development, frequent releases of software, direct stakeholder involvement, minimal documentation and welcome changing requirements even late in the development cycle.* The Agile Requirements Engineering applies the above mentioned principles to the Requirements Engineering process.

Agile Requirements Engineering welcomes changing requirements even late in the development cycle. This is achieved by using the agile practice of e*volutionary requirements* which suggests that requirements should evolve over the course of many iterations rather than being gathered and specified upfront. Hence, changes to requirements even late in the development cycle can be accommodated easily.

There is however, no real process to the agile approach to Requirements Engineering. In order to overcome this disadvantage, we propose to adapt the Requirements Generation Model (a plan-driven Requirements Engineering model) to an agile environment in order to structure the Agile Requirements Engineering process. The hybrid model named the **Agile Requirements Generation Model** is a soft-structured process that supports the intents of the agile approach. This model combines the best features of the Requirements Generation Model and Agile Software Development.

“The woods are lovely, dark and deep.
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep.”

- *Robert Frost*

To

Amma, Appa & Shria

Acknowledgements

I am very grateful to Dr. Arthur, my advisor, for his guidance and support through the two years of my graduate studies. He is the best teacher I have ever had. I have learnt a lot from him these two years. I consider it an honor to have worked with him.

I thank my committee members, Dr. Gröner and Dr. Bohner for all their help, valuable suggestions and a highly memorable thesis defense session.

My parents and my sister are my lifeline. Their unconditional love and support have guided me through times of happiness and disappointments. All that I have achieved so far in my life is due to them. My dream of pursuing a graduate degree would not have become a reality without their encouragement.

Dr. Ahmed Sidky, Dr. Ramya Ravichandar and Amine Chigani, my colleagues in the Software Engineering Research Group have always encouraged and supported me. Amine is my mentor here at Virginia Tech. I am always grateful for his help and guidance.

My roommates Aparna Mahadevan and Anusha Kashyap have been my family here. We have had eventful two years together and these are easily the most memorable ones in my life.

I also thank all my other friends and members of the Indian Student Association here at Virginia Tech for their support.

Table of Contents

List of Figures

List of Tables

1. Introduction

This thesis presents the **Agile Requirements Generation Model** (**Agile RGM**) which is designed to guide and assist the Agile Requirements Engineering (Agile RE) process. It embraces change and adds structure to the Agile RE process by adapting the Requirements Generation Model (RGM) [12] to an agile environment.

The main objective of Agile RE is to accommodate changing requirements even late in the development lifecycle. The Agile RE philosophy is prevalent among organizations and teams that deal with rapidly changing requirements due to changes in software and business environments, evolving user (the person who would actually use the system) needs, technological advancements, etc. However, there is no structure to the Agile RE process. The specification of activities in Agile RE is minimal and there is no mapping between the activities and the techniques that can be used to carry out these activities. This issue can be resolved by structuring the Agile RE process and specifying the activities that have to be carried out.

The RGM is a structured approach to capturing requirements. It covers all the activities of the RE process (requirements elicitation, analysis, specification, verification and validation). It provides guidelines and protocols that can be adopted for the RE phase of the development lifecycle. Its emphasis is on customer (the client who will pay for building the software system) involvement throughout the RE process. The model is flexible and adaptable to an agile environment.

The Agile RGM is motivated by the need to address the insufficiencies identified in the Agile RE process. It structures the Agile RE process. As mentioned earlier, the RGM is adapted to the agile environment to add structure to the Agile RE process. The Agile RGM describes in detail the activities in the Agile RE process and suggests techniques that can be used to perform these activities.

This chapter presents an overview of the agile approach to software engineering and, in particular, to requirements engineering. It discusses in depth, the motivation for this research, the issues involved and the solution approach taken.

1.1 Moving towards agility

Currently, the number of organizations adopting Agile Practices is increasing; some of the reasons being *(1) improved quality, (2) greater return on investment, (3) shorter time to market, (4) enhanced customer relationships, (5) better team morale* [1]. There has been a shifting focus from conventional software engineering approaches towards agility. This section discusses the growth of the agile philosophy and methods.

"Software engineering is (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software (2) The study of approaches in (1)"[2]. The Software Development Lifecycle (SDLC) was introduced in order to structure the software development process and ensure that it complies with the above definition. The SDLC identifies five phases in the software development process: Requirements Analysis, Design, Implementation, Integration and Testing (I&T), and Maintenance [3]. Hence, the software development process systematically begins with identifying and specifying user requirements and then proceeds to design, coding, testing and maintenance. These phases are found in the traditional approaches to software development such as the Waterfall model [3] and the Spiral model [4]. These structured approaches have helped reduce the chaotic state of software development by emphasizing

- Extensive planning to outline the process to be followed, identifying tasks to be completed during product development and determining product milestones
- Comprehensive documentation in the form of Software Requirements Specification (SRS), high- and low-level design documents, test plans, etc.
- Gathering and specifying user requirements upfront before proceeding to the design phase
- Anticipating future requirements and designing the architecture of the system to accommodate current and future requirements
- Process monitoring and control, risk management and software quality control and assurance

However, in the last few years, many development teams have realized that these traditional approaches to software development are onerous and are not suitable to all [5]. The traditional approaches were found to be inadequate and expensive for development efforts which deal with rapidly changing requirements. Conventional methods attempt to foresee future requirements and create the software system architecture upfront in order to accommodate current and future requirements. However, when new requirements that were not previously identified surface, the current architecture may no longer be valid. The cost of modifying the architecture, and in turn the code, in order to accommodate requirements identified late during the development lifecycle is very high [4].

Creating comprehensive documentation is often a wasted activity in the face of changing requirements. Documents have to be maintained regularly to reflect changes made to requirements and design. Hence, in the face of rapidly changing requirements, maintaining comprehensive documentation is expensive. Also, the extensive planning and documentation efforts were found to be cumbersome for organizations and teams involved in developing small scale systems.

Hence, there was a need to develop new cost-effective, light weight methods to accommodate rapidly changing requirements in software systems. This realization motivated the development of the "**Agile**" approach which is best described by the "Manifesto for Agile Software Development" [6] as given below:

> "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"

The core values of the agile manifesto are briefly discussed below:

1. The agile movement focuses on fostering team spirit among the members of the software development team by emphasizing on close team relationships and close working environments. Human aspects of the software development process are considered more important than the process itself.

2. 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.

3. Relationships between the customers and the development team are given preference over strict contracts. Agile methods emphasize delivering software that would provides maximum business value to the customers and hence, reducing the risk of not fulfilling the contract. Also, the customer is involved throughout the development process which fosters better customer-developer relationships.

4. The customers and the development team should be prepared to modify plans in order to accommodate change even late in the development lifecycle.

Agile methods are lightweight processes that focus on *short iterative cycles, direct customer and user involvement, incremental development and accommodating change even late in the development lifecycle*.

In the field of software development, there has been shifting focus from conventional software engineering approaches towards agility. The realization that the traditional plan-driven approaches to software engineering are not suitable to all organizations and teams led to the creation of the Agile Manifesto. The creation of the manifesto gave rise to many agile methods like eXtreme Programming (XP) [14], Scrum [17], Crystal methodologies [15], Feature Driven Development [16] and Lean Development [25], etc. These methods focus on iterative and incremental development, direct customer involvement and accommodating change. The time to develop a product is divided into multiple release cycles and each release cycle comprises of multiple iterations. During each release cycle, a potentially shippable increment of the product is developed. Hence, the product is developed in an iterative and incremental fashion. Agile methods focus on direct customer and user involvement. The customers and users are involved through the complete product development lifecycle.

1.2 Agile RE

The main objective of Agile RE is to accommodate changing requirements even late in the development lifecycle. Agile RE applies the focal values and principles mentioned in the agile manifesto to the RE process. This section briefly describes the importance of RE as an activity in the SDLC, the issues with the conventional approaches to RE and the Agile RE. RE is the most important activity in the **S**oftware **D**evelopment **L**ife **C**ycle (**SDLC**) as a system is only as good as its requirements. All the other phases of the SDLC depend on the requirements engineering phase. According to the CHAOS report published by the Standish Group in 1995 [7], a clear statement of requirements is one of the three most important factors for a project to succeed. The significance of requirements is best expressed by F.P. Brooks [8]

> "The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later"

Studies have shown that the cost of correcting faulty requirements later during the coding phase in the SDLC is up to 200 times more than correction during the requirements phase [4]. These studies mention that a set of complete and unambiguous requirements is essential to the success of a project.

The requirements engineering phase of the SDLC comprises of the following activities:

- **Requirements Elicitation –** Discover, understand, articulate and record customer and user needs

- **Requirements Analysis –** Understand needs/constraints of the user and derive the set of requirements
- **Requirements Verification –** Ensure that the requirements are adequate and conform to agreed standards
- **Requirements Specification –** Record the requirements in an unambiguous manner
- **Requirements Management –** Plan and Control the above activities

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 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 [9]:

- *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.
- *Developers* and *requirements engineers* often *misinterpret or fail to capture customer and user needs and intents* and thereby, specify incorrect requirements.
- *Customers and users* are not very clear about their needs and often state the desired functionality in terms that can mislead or result in incorrect inferences by the requirements engineer.
- Rapidly changing business environments can cause requirements to become *obsolete* before project completion.

These issues have caused organizations involved in software development to challenge the conventional RE practices. In order to overcome the problems caused by following the traditional RE methods, Agile RE is adopted. The Agile RE process accommodates changes to requirements even late in the development lifecycle. This is achieved by applying the practice of evolutionary requirements which suggests that requirements should evolve over time.

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.* The agile approach to RE applies the above mentioned principles to the RE process. Agile RE as a solution to the issues faced by practitioners adopting conventional RE approaches, is discussed briefly below.

- Agile RE [10] 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 iteratively and incrementally rather than being gathered and specified up front. Hence, changes to requirements can be accommodated easily. Initially, the high level features for the system are defined where features indicate the expected functionality. These high-level features help identify the scope of the system. Further details for the features in the form of requirements or stories are gathered **j**ust-**i**n-**t**ime (**JIT**) from the customers before the development of that feature. This practice of not gathering all the requirements upfront is known as No **B**ig **R**equirements **U**p **F**ront (**BRUF**). As Agile RE adopts No BRUF and suggests gathering details in a JIT fashion, changes made to the features under consideration for implementation during a release cycle, do not affect the other features identified earlier. Hence, changing requirements are accommodated easily. Usually, changes to requirements are logged and are implemented during the current or in the following iterations.

- Maintaining comprehensive documentation is an important activity specified by all conventional RE methods. However, as mentioned earlier, maintaining comprehensive documentation is expensive in the face of changing requirements. Agile methods address this issue by focusing 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.

The main issue faced by practitioners adopting conventional RE methods is accommodating changing requirements. The Agile RE process which accommodates changing requirements is being adopted by organizations in order to address the issues discussed earlier.

1.3 Motivation for the Agile RGM

Agile RE has gained popularity as it helps accommodate change even late in the development lifecycle. However, knowledge of Agile RE is mostly tacit and there is no formal or structured approach to it which would guide practitioners adopting it. This research is motivated by the need to address the lack of a structured approach to Agile RE.

Organizations and teams adopt Agile RE in order to accommodate rapidly changing requirements. The major Agile RE process activities can be summarized as below:

1. Identify the high level features of the system. The features denote the functionalities expected by the customers.
2. Prioritize the features based on the customer needs and their business value.
3. For each feature to be developed, gather the details JIT from the customers and users.
4. With the gathered details for the features, proceed to development and acceptance testing.

Though the idea is prevalent, knowledge of the Agile RE activities and techniques to carry out the activities is mostly tacit. The specification of activities is minimal and there is no comprehensive set of techniques that practitioners can choose from to perform these activities. The Agile RE process is not structured. Hence, if organizations or teams wish to adopt Agile RE, there is no approach that would guide them through the process. Thus, there is a need to outline an approach which would clearly delineate the activities in Agile RE and suggest techniques that can be used. Such an approach would serve as a guideline to the Agile RE process. Hence, the approach should reflect the Agile RE philosophy and add some structure to the process in order to better guide practitioners. The approach should be soft-structured in that while attempting to add structure to the Agile RE process, the impact on agility should be minimal.

Scott Ambler's model for Agile RE [10] reflects the agile approach to RE but does not give a step-by-step guideline for practitioners to follow. Also, it does not give suggestions about the techniques that can be used. Conventional RE models such as Requirements Triage [11], Requirements Generation Model [12] and Win-Win Spiral Model [13] describe in length the activities of the RE phase and suggest practices and techniques that can be adopted for each phase. Such specification of activities is minimal in Scott Ambler's model and there is no mention of the methods that could be used. There is minimal mention of RE activities and practices that can be followed in the existing agile methods which can confuse practitioners as one of the most important issues faced by development teams is the identification of techniques that can be used to serve their purpose.

Documentation review is an important activity suggested by conventional RE approaches in order to verify and validate the requirements and ensure that the gathered requirements are complete. Agile methods insist on minimal documentation. In the absence of comprehensive documentation, there is a need to ensure that the RE process is complete.

Hence, there is a need for a structured approach that would serve as a guideline to the Agile RE process that explicitly defines the activities and suggests techniques that can be used for each activity. This structured approach should embrace agility.

1.4 Problem Statement

The main objective for our research is *to develop a soft-structured approach for Agile RE*. As mentioned in the previous section, clear specification of activities in the Agile RE process is missing and there is a lack of a set of techniques that practitioners can choose from. There is no structure to the Agile RE process. Hence, there is a need to develop a structured approach that clearly outlines the activities of the Agile RE process and suggests techniques or practices that can be used. Such an approach guides practitioners adopting Agile RE. To achieve clear specification of activities in the Agile RE process, it must be structured. However, it is imperative that the impact on agility is minimal. Hence, the new approach should be soft-structured in that it structures the Agile RE process without compromising agility.

This research creates a soft structured approach to Agile RE which reflects the Agile RE philosophy and adds some structure to the process with minimal impact on agility. The issues involved in solving the identified problem are discussed in the next paragraph.

Attempting to add structure to the Agile RE process with minimal impact on agility raises a few issues due to the conflicting philosophies of structured software development approaches and agile methods. However, the following issues need to be addressed in the solution approach. The following are the issues to be resolved:

- The *target systems* should be identified for this model based on their size, complexity and time for completion. Agile methods are not considered suitable for the development of large scale and Mission and Life Critical (MLC) systems [1]. Development of large scale systems span multiple (more than two) years. Failure of MLC systems can cause destruction to property and life. The agile practices of evolutionary requirements and refactoring are not suitable for developing mission and life critical systems. The reasons for questioning the suitability of agile methods for large scale systems development is discussed in Chapter 5.

- Agilists believe in *minimal* or *no documentation* and *no BRUF*. The model should account for these principles. On the other hand, structured approaches to software development emphasize on comprehensive documentation and upfront requirements gathering and specification efforts.

- The solution should meet the *intents* of the agile approach as specified in the agile manifesto.

- The idea of *needs/ requirements* in the traditional approach to RE should be reconciled with *features/ stories* in the agile approach to RE.

- The solution should be developed within a framework of the development process.

The activities of Agile RE are not specified clearly. Also, there is no existing mapping between the activities and the techniques that can be used to carry out these activities. Hence, there is a need for a structured approach to Agile RE. The proposed solution should address the above mentioned issues. The next section describes the solution approach taken.

1.5 Solution Approach

The Agile RGM presented in this thesis is a soft-structured approach which structures the Agile RE process without compromising agility. This approach clearly specifies the activities in the Agile RE process and suggests techniques that can be used. It guides practitioners adopting Agile RE. This solution approach discussed in this section, consists of two major phases which are

- Phase 1: Identification of a structured RE model that can be adapted to reflect the agile approach to RE
- Phase 2: Modifying the model to reflect agility

Phase 1: The first step in the solution approach is to identify a traditional RE model which can be adapted to reflect the intents of the agile approach. The RGM [12] is a structured approach to capturing requirements. It includes all the major RE phases of requirements elicitation, analysis, specification, verification and management. It provides guidelines and protocols for requirements gathering that practitioners must adopt.

Agile methods suggest iterative and incremental development and direct customer involvement is considered essential. The RGM is an iterative model and focuses on customer involvement. These characteristics are reflective of the agile philosophy. Also, the RGM provides clear specification of activities with adequate scope for modification. Hence, the RGM is chosen to structure the Agile RE process. The RGM is adapted to reflect agility.

Phase 2: The next step is to modify the RGM to reflect the Agile RE philosophy such that its structure is preserved and the impact on agility is minimal. This can be achieved by modifying the RGM to embrace agility. The structure of the RGM must be preserved. However, the activities and techniques suggested by the RGM may be modified in order to adapt the RGM to an agile environment. Then, the various phases of the model, the activities that are carried out in each phase and techniques that can be used are specified.

The solution approach suggested in this section involves the identification of a traditional RE approach that can be adapted to an agile environment and specifying activities and techniques that can be used by practitioners adopting Agile RE. In this case, the RGM is modified to suit the agile environment. Using the RGM, the Agile RE process is structured.

1.6 Blueprint

The next chapter, Chapter 2 presents background information about agile methods, Agile RE concepts and the RGM. The Agile RGM which was designed by adapting the RGM to an agile environment is discussed in Chapter 3. The Agile RGM is an independent model that can be used with any of the existing agile methods like XP, Scrum, etc. Substantiation of this claim is discussed in Chapter 4. The Agile RGM has been designed for small scale systems with development periods of less than one year. However, the Agile RGM can be further modified to accommodate large scale systems development. Chapter 5 suggests an overview of an alternative structure to the Agile RGM which can be used for the development of large scale systems. Finally, Chapter 6 provides concluding remarks about the Agile RGM and suggests future research activities related to the Agile RGM.

2. Background

The solution approach described in the previous chapter identifies a soft structured approach to agile RE process by adapting the Requirements Generation Model (RGM) [12] to reflect agility. The objective of this chapter is to briefly describe agile principles, concepts and methods, agile RE principles, concepts and methods, and the RGM which serve as the basis 6or the Agile RGM model to be discussed in the next chapter.

2.1 Agile Principles

The agile approach is best described by the "Manifesto for Agile Software Development" which is given below [6]:

> "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.

The principles behind the agile manifesto are stated below:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly"

The agile approach to software development reflects the focal values and the above mentioned principles as stated in the agile manifesto. Agile approach to software development focuses on iterative and incremental development, evolutionary requirements, code refactoring, just-in-time gathering of details, no BRUF and direct customer involvement. Accommodating change even late in the development lifecycle is one of the most important principles of the agile approach to software development which is achieved by advocating practices such as evolutionary requirements and just-in-time philosophy.

Customer satisfaction is of the highest priority to the development teams. The teams strive to satisfy the customers and users through early and continuous delivery of software that is of value to them. The teams deliver software frequently every two weeks to every month in order to obtain customer and user feedback. As the customers are involved in every step of the process, changes to needs and requirements are identified quickly.

The agile approach places importance on people and their interactions rather than the processes and tools used for development. Hence, teams adopting the agile approach propose open environments for working, encourage face-to-face communication among the stakeholders (stakeholders include customers, users, developers, project managers, business analysts, testers, etc.) and provide support to the team members. The stakeholders work together throughout the project.

Many agile methods such as eXtreme Programming (XP) [14], Scrum [17], Crystal Methods [15], Feature Driven Development [16], Dynamic Systems Development [15], Lean Development [25], etc., were brought into light after the creation of the agile manifesto. A brief description of some of these methods is given in the next section.

2.2 Agile Methods

Organizations and teams found that the traditional software engineering approaches like the waterfall model are onerous especially for small scale projects and are inadequate for projects facing rapidly changing requirements. This realization led to the creation of the agile manifesto and subsequently, the agile methods came into existence. Most of the agile methods stem from the best practices in software development and in industry used by teams and organizations over many years. For example, Lean software development [25], one of the existing agile methods, has its origin from the Lean Principles applied in the automobile industry. This section briefly describes some of the agile development methods in practice.

2.2.1 eXtreme Programming (XP)

XP is designed for small to medium teams which have to develop software quickly in the face of rapidly changing requirements. XP came into existence with software development practices found effective during the previous decades. The XP methodology was developed after a number of successful trials of using the key practices and principles identified. XP is based on 4 values: communication, simplicity, feedback and courage [14]. “It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation” [14]. The customers are actively involved in the development process.

The stakeholders meet to define the scope of the system and identify features based on their business value. The identified features are prioritized and the feature set to be implemented during the next release is determined. The design of the system is created and is usually kept simple. The development team then proceeds to implement the features using Test Driven Development (TDD). The customers then test the system to determine if their needs are addressed. Some of the XP practices [14] are discussed briefly below:

1. **The Planning Process or the Planning Game –** Customers prioritize the identified features based on their business value and decide on the feature set to be implemented next (Release planning). Also, the work to be done next is determined (Iteration planning).

2. **Small Releases –** Teams build and deploy the system in small increments and update the deployed parts during subsequent iterations.

3. **Simple Design –** Teams spend less time on design and keep it simple. The focus is to design the simplest solution which would provide maximum business value to the customers. The design is constantly refactored in order to accommodate change.

4. **Test Driven Development –** Programmers write tests first and then develop code only when the tests fail.

5. **Refactoring –** This practice is used to restructure the system in order to improve the design and code. Refactoring is achieved by removing duplication, increasing cohesion and lowering coupling of code. Refactoring helps accommodate changing requirements even late in the development lifecycle.

6. **Pair Programming –** Two programmers work together at one machine writing code and performing code review simultaneously.

7. **Collective Ownership –** The code produced belongs to the entire team and hence anyone can change the code when required.

8. **Continuous Integration –** When a new piece of code is developed, it is integrated into the existing code-base. The system is integrated multiple times each day.

9. **On-site Customer –** A dedicated individual (customer representative) is available at the development site to determine requirements, set priorities and answer questions as and when required.

10. **Customer Acceptance Tests –** The customers create acceptance criteria for each feature and the system is validated based on these criteria.

The process begins with the Planning Game. The stakeholders identify and prioritize the features for the system to be built. The feature set to be implemented during the next release is determined. The work to be completed during the next iteration is also determined. Each iteration lasts for about 2 – 4 weeks. The development team then implements the features using TDD and the customers run the acceptance tests. The process continues till the product is complete.

XP is a set of practices that can be adopted by agile practitioners. The practitioners can choose to use some or all of the practices. However, XP does not provide structure to the development process. Hence, development teams often adopt Scrum which provides a process management framework within which XP practices can be used. Scrum is discussed in section 2.2.3. As mentioned earlier, XP is designed for small teams. However, XP practices can be used in large scale systems development as well. For example, pair programming can be adopted by teams involved in large scale systems development.

Shortcomings of XP

By design, XP has the following shortcomings:

1. No structure to the Agile RE process

2. Process management activities are not specified in detail

3. TDD and refactoring are considered unsuitable for large scale systems development

2.2.2 Feature Driven Development (FDD)

Feature Driven Development [15] [16] is an agile approach to developing software systems. FDD does not cover the entire software development process. It focuses on the design and coding phases. However, it can be used in conjunction with the other activities in the software development process. FDD follows an iterative approach and uses the best practices found in the industry.

FDD consists of five processes which are described below:

Process 1: **Develop an overall model –** Documented requirements in the form of use cases or functional specifications serve as an input to the FDD process. The domain experts utilize the documented requirements to develop a high level scope of the system and provide a walkthrough to the other stakeholders. The objective is to provide "the big picture" of the system to the stakeholders. The overall domain is then further divided into different domain areas and detailed walkthroughs are held. During the walkthroughs, object models are produced for each of the identified domains.

Process 2: Build a features list – The documented requirements and object models provide a good basis for building a comprehensive features list for the system. The features are client valued functions for each of the identified domain areas. The functions grouped according to the domain areas form the feature sets. The feature lists are reviewed by customers and users for their validity and completeness.

Process 3: Plan by feature – A high-level plan is produced and the feature sets are sequenced based on their priorities and dependencies. Schedule and major milestones may be set.

Processes 4 and 5: Design by Feature and Build by Feature – These two processes are iterative. The set of features to be developed over the next few days (no longer than 2 weeks) is identified. Feature teams comprising of class owners needed for building the feature sets are formed. Each feature team produces detailed sequence diagrams for the features and writes class and method prologues [16]. The team then conducts a design inspection. When the inspection is successful, class owners write code, perform unit test, integrate and conduct a code inspection. The completed features are then added to the existing code base. The design by feature and build by feature processes are repeated for other feature sets.

FDD does not address the process of gathering requirements. It focuses only on the design and coding phases of development. As it does not span the complete development process, other methods should be used in order to address the issue of eliciting requirements.

Shortcomings of FDD

By design, FDD does not include RE process activities. Some form of requirements documentation is given as input to the FDD process. The following shortcomings of FDD with respect to Agile RE are due to its design.

1. No specification of RE process activities
2. Focuses only on the design and coding phases of software development and does not span the complete development process.
3. Requires other process approaches in order to support other phases of software development.

2.2.3 Scrum

Scrum [15] [17] is an agile approach developed to manage the software development process. Scrum employs an iterative and incremental process skeleton that includes a set of pre-determined practices and roles. It introduces flexibility, adaptability and productivity to the software development process [15]. However, it does not specify implementation techniques. The product is built in small increments. Work is structured in iterations called sprints lasting for 2 - 4 weeks. At the end of each sprint, a potentially shippable product increment is produced.

The Scrum process starts with the *Sprint planning meeting*. Initially, the customer produces the *product backlog* which is a set of prioritized features which have the maximum business value. Planning for the sprint begins with choosing the features that would be developed during the current sprint and the expected work result. After the feature set is determined, each feature is broken down into *sprint tasks* which are the development activities required to implement a feature. The sprint tasks form the *sprint backlog*.

After planning for the sprint, the sprint begins its iteration cycle. Each day, the team meets for about fifteen minutes where each member of the team answers the following three questions:

"*what did I do yesterday, what did I do today, and what impediments got in my way?*" [15].This meeting is called a *Daily Scrum Meeting*. The objective of this meeting is to address needs of the members of the team, adjust the work plan and discover new dependencies.

At the end of each sprint, a *Sprint review meeting* is convened. The objective is to demonstrate the potentially shippable product increment and to determine if the features have been completed. Validation of the product increment is carried out by the product owner.

As mentioned earlier, Scrum does not suggest development activities and implementation techniques. It provides a framework for process management. Hence, practitioners adopt other agile methods like XP and FDD that specify the activities to be carried out and provide implementation techniques. For example, a development team adopting Scrum can use TDD to implement the identified features.

Shortcomings of Scrum

Scrum has been designed as a process management framework. Its shortcomings listed below with respect to Agile RE and Agile development are due to the design of the process.

1. Does not specify implementation techniques
2. Requires practitioners to adopt other agile methods to specify the development process activities

2.3 Agile RE

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.* The agile approach to requirements engineering applies the above mentioned principles to the RE process and is described in this section.

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, the Agile RE [10] 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. 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. Customers 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.

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. Figure 2.1 below describes the Agile RE process [10].

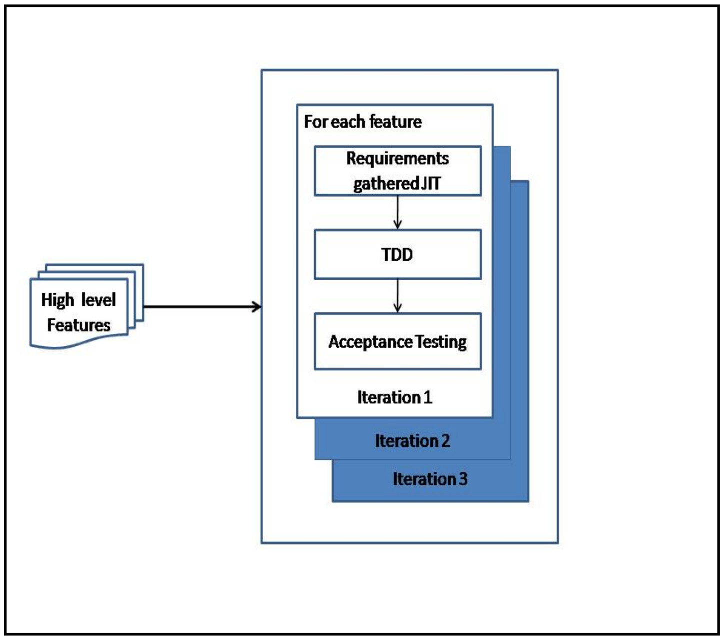


Figure 2.1.Agile RE Process

The steps in the Agile RE process are:

1. Identify the high level features of the system.

2. Prioritize the features.

3. For each feature to be developed, gather the details JIT from the customers.

4. With the gathered details, proceed to development (**TDD**) and acceptance testing.

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.

2.3.1 Differences between Agile RE and Conventional RE

This section discusses the differences between Agile RE and Conventional RE. The most significant difference between Agile RE and Conventional RE is that Agile RE emphasizes *evolutionary requirements*, *no BRUF* and the requirements are gathered just-in-time.
Also, Agile RE focuses on *minimal requirements documentation* as opposed to the formal SRS document produced with the Conventional RE approach. Traditional RE models define explicit Requirements Verification and Validation activities. *Verification and Validation of requirements* is done implicitly and just-in-time with Agile RE. The stakeholders can view the requirements recorded on index cards anytime and can verify and validate the requirements. Verification of

requirements using the Agile RE approach is more of a validation process. Table 2.1 specifies the differences between Agile RE and Conventional RE.

Table 2.1.Differences between Agile RE and Conventional RE

	Agile RE	**Conventional RE**
1.	Evolutionary requirements. No BRUF. Requirements are gathered just-in-time.	Requirements do not evolve over time. All the requirements are gathered up front before the design phase.
2.	Minimal requirements documentation. Requirements are stored on index cards.	Formal Software Requirements Specification (SRS) document is produced.
3.	Implicit verification and validation of requirements. V&V is carried out just-in-time.	Explicit verification and validation of requirements activities.
4.	Changes to requirements even late in the development are accommodated.	Changes in requirements during the later phases of the development life cycle are not accommodated.
5.	The time for project completion is decomposed into multiple release cycles and the features are estimated to fit within these release cycles.	Requirements are gathered and specified before estimating the time required to develop these requirements.
6.	On-site customer is present to answer questions from the development team	No on-site customers.
7.	Requirements are recorded on index cards in the domain language of the user.	Requirements are specified in a formal language

2.3.2 Advantages and Disadvantages of Agile RE

The main advantage of Agile RE lies in the fact that it can accommodate changing requirements even late in the development lifecycle. This reduces the cost of accommodating change. However, there is no real process to the Agile RE approach. There is no specification of activities to be carried out and techniques that can be adopted. The advantages and disadvantages of Agile RE are summarized below.

Advantages:

- Accommodates changing requirements even late in the development cycle
- Mitigates issues faced when gathering requirements upfront as discussed in Chapter 1.

Disadvantages:

- No real process/ specification of activities in the RE phase
- Non-functional requirements are not well-defined. The focus is on identifying all the requirements and not on classifying the requirements into functional and non-functional requirements. Also, non-functional requirements are recorded as user stories.

2.4 Agile Requirements Modeling

This section describes the agile requirements modeling described in Scott Ambler's **Agile Model Driven Development (AMDD)** lifecycle [10].

The *initial requirements modeling* and *model storming* are aspects of Agile RE. The fundamental idea is to do enough modeling at the start of the project in order to understand the system requirements at the highest level and then gather details as and when required in a just-in-time fashion. These two phases relevant to the Agile RE process and are described next.

Initial Requirements Modeling

At the beginning of the project, just enough modeling should be done in order to get the "big picture". Initial requirements modeling helps identify the scope of the system. The initial requirements model may take the form of use cases, user stories, UML class diagrams or user interface prototypes. These requirements artifacts provide just enough information to help understand the system at the highest level.

Model Storming

The high level requirements identified during the initial requirements modeling phase are analyzed. This involves gathering details in a just-in-time fashion. The development team request customers to provide information about the gathered requirements. The details are gathered for one requirement or a subset of the identified requirements at a time.

Shortcomings of AMDD

1. The model does not provide a clear specification of the activities to be carried out when adopting the Agile RE approach.
2. There is no mention of the techniques that can be used for gathering the initial requirements.

2.5 Requirements Generation Model (RGM)

The RGM [12] is a structured approach to capturing requirements. The RGM covers all the activities of the requirements engineering process namely requirements elicitation, analysis, specification, verification and management. “The RGM is based on two components, 1) a ***framework*** that structures and controls the activities within which the customer and the requirements engineer should define requirements, and 2) a ***monitoring methodology*** that ensures that all requirements elicitation activities follow proper procedures”[12]. The framework consists of interdependent phases and a set of structuring components supporting the requirements generation process. Guidelines, protocols and the monitoring methodology form the set of structuring components that direct activities within and among the various phases [12].

The **Monitoring Methodology** continuously applies procedures to monitor activities within the requirements elicitation process to detect irregularities and correct them. Figure 2.2 shows the RGM.

The phases in the RGM are described below:

1. **Indoctrination phase** - governs all preparation activities up to and including the initial requirements elicitation meeting between the customer and the requirements engineer. The objectives of this phase are to educate the customers about the RGM and to ensure that the developers learn the business process of the customers, their problem domain and needs [12].

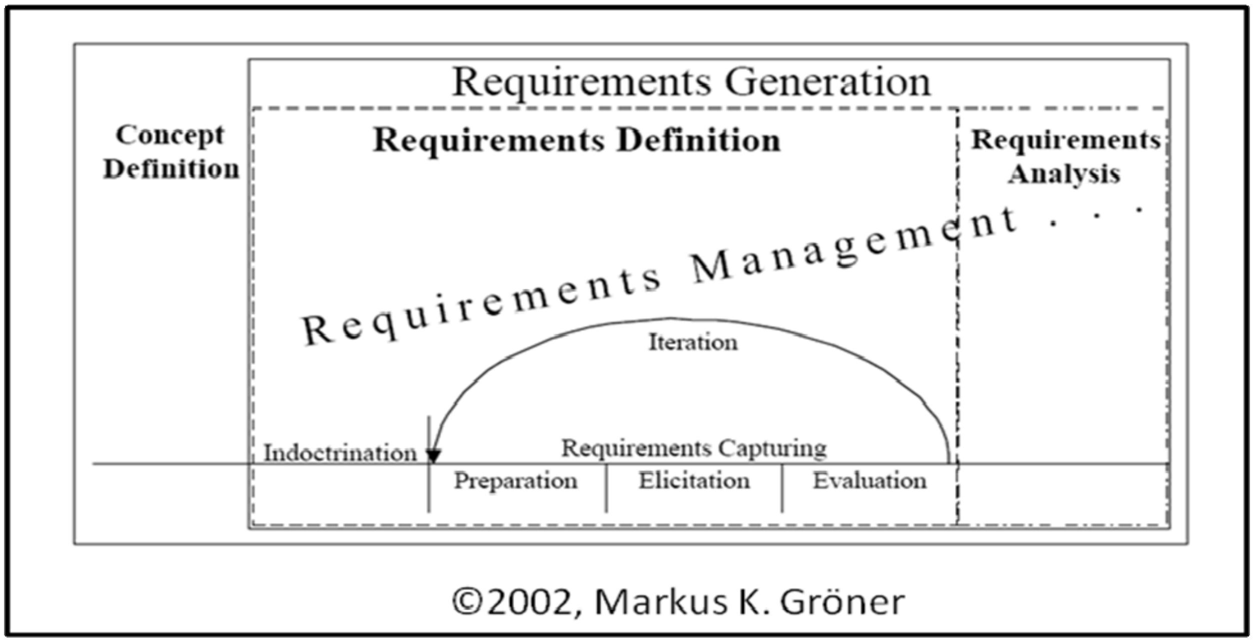


Figure 2.2.The Requirements Generation Framework [12]

2. **Requirements Capturing phase –** This phase involves the three activities of preparation, elicitation and evaluation.

   - **Preparation** – scope of the elicitation meeting is defined and identified issues are resolved.
   - **Elicitation** – requirements are identified and recorded accurately
   - **Evaluation** – identified requirements are reviewed, new or unresolved issues are identified and the need for additional iterations of the requirements capturing phase is determined

Requirements management pervades the requirements generation process.

Advantages and Incompatibilities with respect to Agile RE

Advantages:

- Structures the RE process
- Includes all the activities of the RE process
- Iterative process

- Encourages collaboration among the stakeholders
- Requirements are produced in increments

The advantages mentioned above reflect agility.

Incompatibilities with respect to Agile RE

- Focuses on BRUF
- Not evolutionary
- Does not accommodate changes to requirements after specification
- Enforces on creating formal documentation

The RGM reflects the agile philosophy. The model is flexible and hence, can be modified to be used in an agile environment.

2.6 Precursor to the Agile RGM

The RGM described in the previous section is a flexible model and reflects certain agile principles. This model uses the RGM for gathering requirements for a project using agile methods. The objective is to create a soft structured process that would preserve the structure of the RGM with minimal impact on agility. Figure 2.3 shows the precursor to the Agile RGM [26].

The process followed is summarized below:

- Using RGM approach, identify major project features. These features denote the functionality expected by the customers and users. Identifying the features up front helps determine the scope of the system to be built.
- Determine high-level architecture
- For each feature, using RGM approach, identify Release-Level Stories (RLS)
- For each RLS, iteratively refine and develop using Test Driven Development

This model advocates the use of just-in-time gathering of details and hence, can accommodate changing requirements even late in the development lifecycle. The development process is also considered in this model.

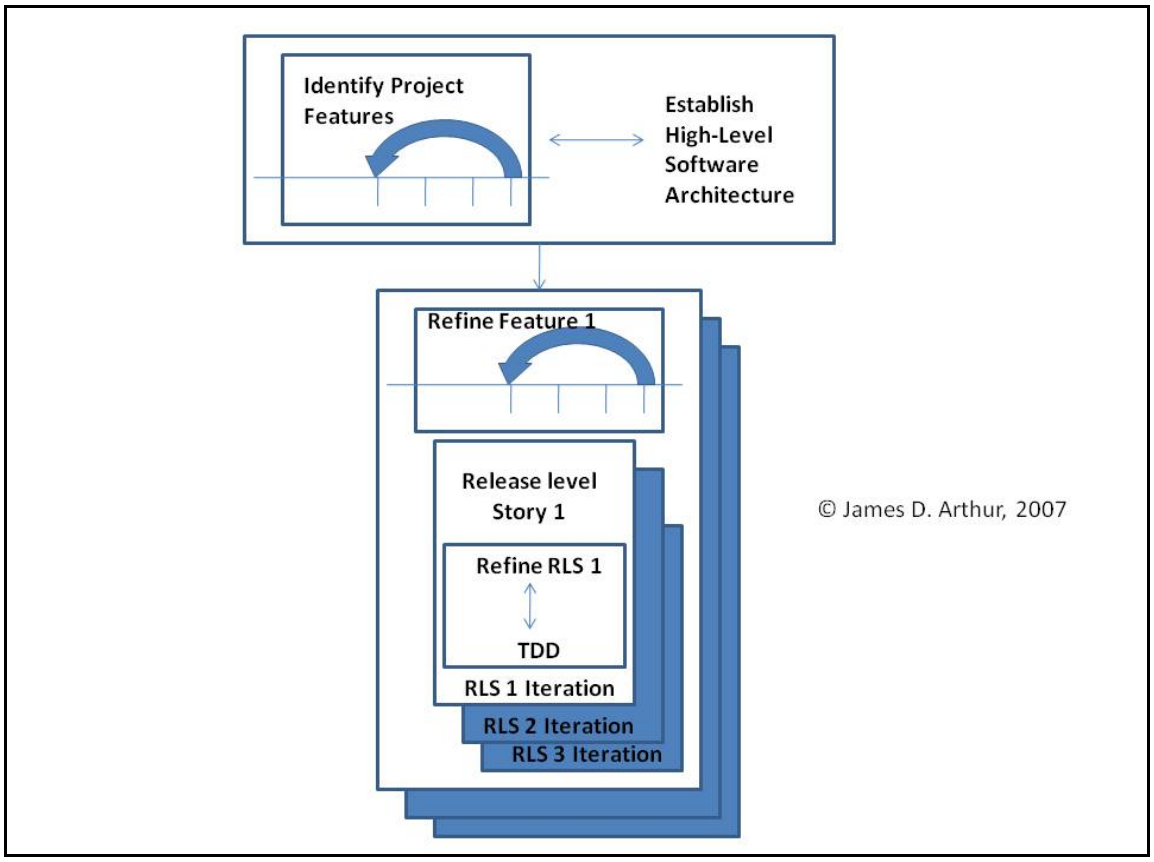


Figure 2.3.Precursor to the Agile RGM [26]

Shortcomings of the Precursor to the Agile RGM

1. There is no specification of the RGM activities that can be adapted to the Agile RE process

2. There is no mention of techniques that can be used

2.7 Summary

This chapter provided background information about the agile philosophy, agile methods and the RGM. The Agile RGM was designed by adapting the RGM to reflect the Agile RE process. Agile RE focuses on applying agile principles and practices to the RE process. The main objective of Agile RE is to accommodate changing requirements. The RGM is reflective of the agile philosophy and can be modified to suit an agile development environment. The next chapter discusses the Agile RGM which is a soft-structured approach to Agile RE.

3. Agile Requirements Generation Model (Agile RGM)

The objective of this chapter is to describe the Agile Requirements Generation Model (Figure 3.1). The need for a soft-structured approach to agile RE was emphasized previously. Agile RGM is designed for short term (having less than one year development period), non- MLC agile projects with 60 to 90 day release cycles. Agile practices such as minimal documentation, refactoring and TDD are not considered suitable for large scale and MLC systems. The Agile RGM is designed to adopt these practices. Hence, the target systems for the Agile RGM are small scale and non-MLC systems. As the length of the development lifecycle is taken into account, the Agile RGM describes not only the RE process activities but the complete development process as well.

The Agile RGM was motivated by the lack of structure to the agile RE process. Creation of the Agile RGM involved adapting the Requirements Generation Model (RGM) [12] to an agile environment and to reflect the agile RE process. The RGM is a structured approach to capturing requirements. It provides guidelines and protocols that practitioners must adopt in order to gather requirements. The structure of the RGM is preserved in the Agile RGM.

The Agile RGM attempts to structure the agile RE process with minimal impact on agility. It reflects the agile principles such as direct stakeholder involvement, evolutionary requirements, refactoring, no BRUF, just-in-time gathering of details and minimal documentation. Verification and validation activities in the existing agile methods are predominantly validation efforts carried out implicitly. The Agile RGM explicitly specifies validation as an activity for each phase.

The Agile RGM consists of five phases. They are

1. **Education Phase –** The objective of this phase is to establish rapport among the various project stakeholders. The development team learns the business process of the customers. Also, a high level mission statement for the project is created which serves as the input to the next phase in the model.

2. **Feature Development Phase –** The stakeholders derive the high level features of the system from the high level mission statement created during the Education Phase. These features are sets of functionality that have business value to the customer. Only one

feature or a subset of the identified features is chosen for development during a release. The identified features are validated, the time for their completion estimated and are prioritized based on their value to the customers. The output of this phase is a prioritized feature stack. During any release cycle, the subset of features chosen for development during the current release will be the input to the Story Development Phase.

3. **Story Development Phase –** Each identified feature is decomposed into stories. The stories are brief descriptions of user- or customer- valued functionality. The stories are validated and the developers estimate the time required for their completion. The stories are implemented during the iterations. Hence, the developers estimate the time required to implement each story in this phase. The stories are then prioritized and stored in a prioritized story stack. A subset of the prioritized stories is chosen for development during an iteration which lasts for about 2 to 4 weeks. This subset serves as the input to the Task Development Phase.

4. **Task Development Phase –** Tasks are essentially checklists for developers which outline the details required for implementing the stories. Each story identified in the previous phase is decomposed into tasks. The developers validate the tasks and estimate the time required to complete the tasks. Each task is estimated to take a few hours to be implemented. These tasks are then developed during the Development Phase.

5. **Development Phase –** The tasks identified in the previous phase are implemented using Test-Driven Development (TDD) approach. The customers and developers perform acceptance testing to ensure that the system meets the customer and user needs.

Figure 3.1 shows the Agile RGM. The model consists of five phases as mentioned above. The first step is the education phase where the objective is to establish rapport among the various stakeholders and create a high level mission statement for the system to be built. This high level statement serves as the input to the Feature Development Phase. The stakeholders iteratively identify the expected system functionality (features) from the mission statement. These features are stated at the highest level of abstraction. The identified features are validated by the customers. The developers estimate the time required for the completion of each feature.

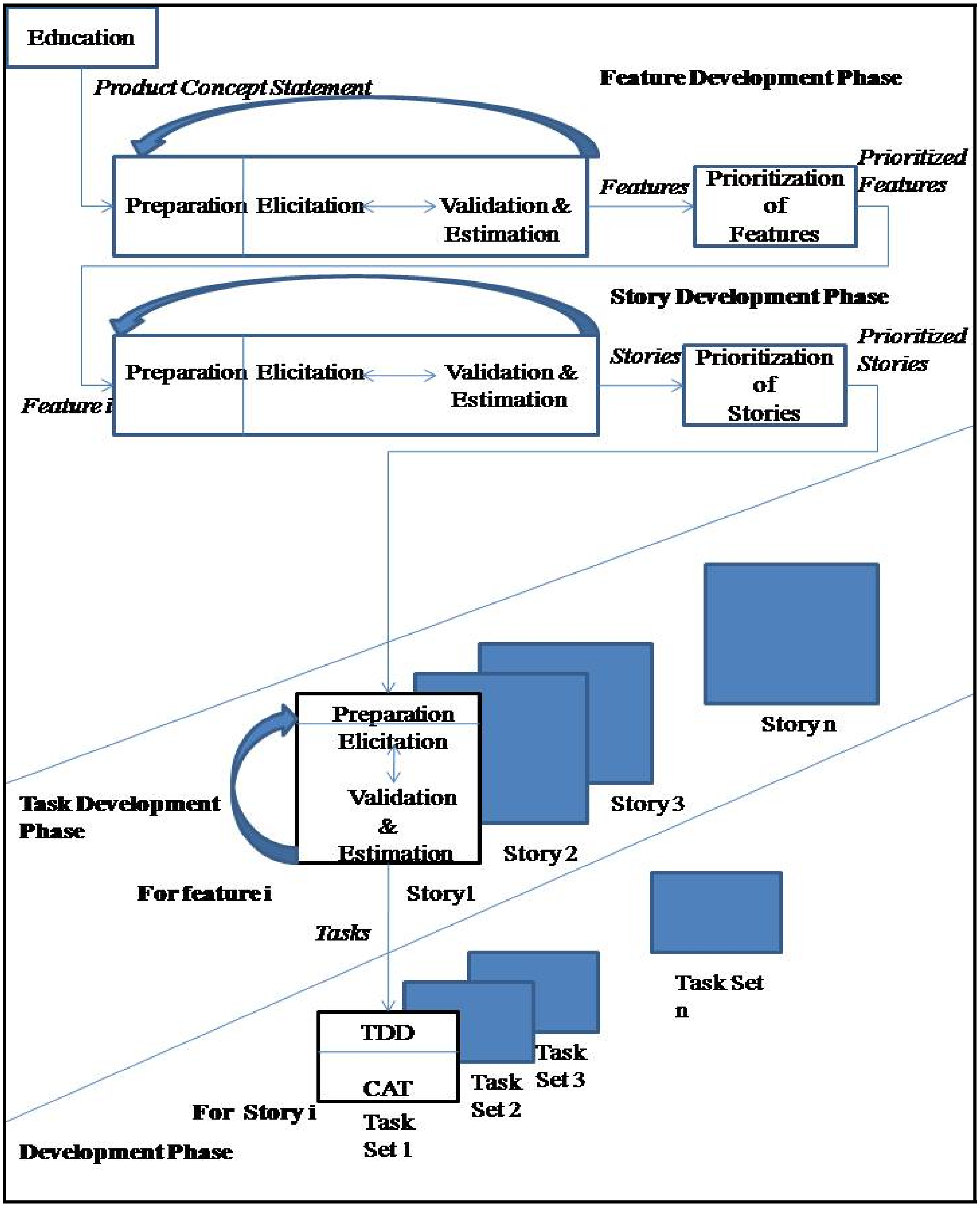


Figure 3.1.Agile RGM

The customers prioritize the identified features based on the business value of each feature. These prioritized features are stored in a stack in the order of their priorities. This stack is known as a prioritized feature stack. Only one feature or a subset of the identified features is chosen for implementation during a release cycle. The details for this feature or subset of features are gathered just-in-time. The remaining features will be implemented during the future release cycles. The stakeholders should strive to identify as many features as possible before proceeding to the next phase in order to be informed of the scope of the system.

Each feature chosen to be implemented during the current release is decomposed into stories. If multiple teams are involved in the development of the system, each team can work towards decomposing one or more features into stories. Stories are user- or customer- expected functionality. The stories for each feature are identified over a number of iterations and are then validated, estimated and prioritized. The prioritized stories are stored in a prioritized story stack. Each story is then decomposed into tasks during the Task Development Phase. The task list for each story is a to-do list for the developers giving the details required to implement the stories. The tasks are then developed using TDD and the customers and developers test execute the acceptance tests. Figure 3.2 shows how each feature is decomposed into multiple stories and each story into multiple tasks.

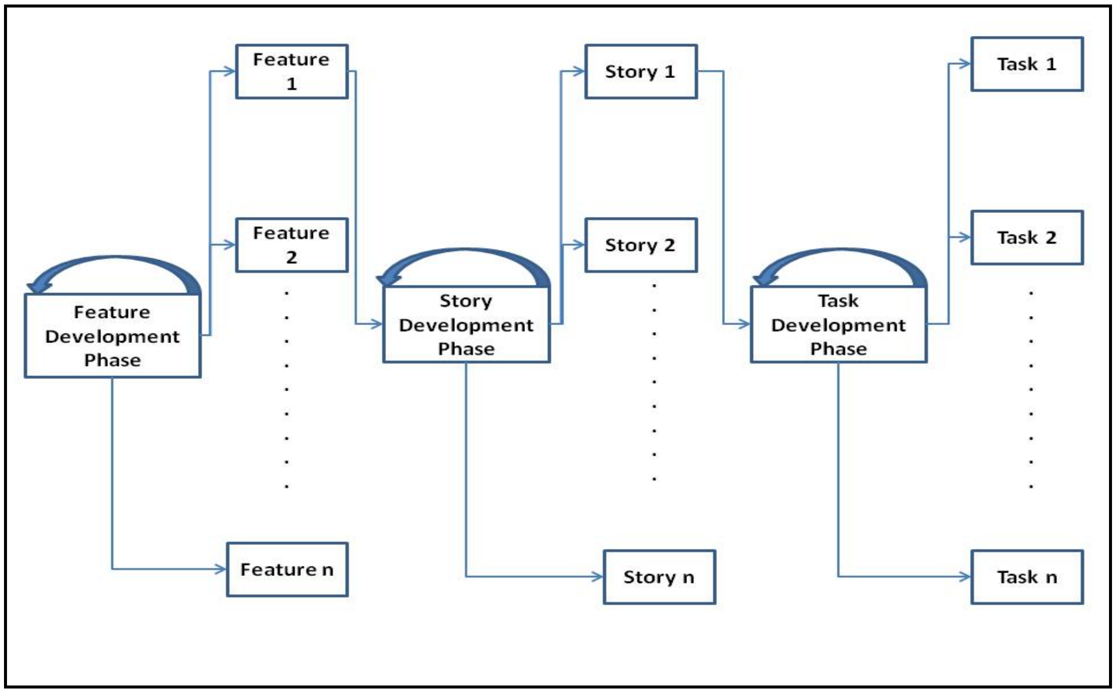


Figure 3.2.Mapping between features, stories and tasks

Verification and validation of features, stories and tasks are not carried out explicitly in Agile RE. As customers are directly involved and are usually available on-site, validation is done just-in-time and implicitly. However, the Agile RGM explicitly defines the validation activity in each phase as shown in Figure 3.1. The validation process is to ensure that the customer and user needs and intents are captured correctly. The validation is carried out just-in-time. Validation is a very important activity in software development process. The Agile RGM includes an explicit validation activity and ensures that it is done in a JIT fashion.

Agile RGM supports concurrency from the story development phase onwards. If multiple teams are involved, each team can write stories for one or more features chosen for development concurrently. Similarly, each team can create task lists for one or more stories and multiple tasks can be implemented in parallel. This high degree of concurrency ensures faster development of the system. However, the disadvantage is that it accommodates the JIT philosophy to very less extent. Features are identified upfront and there is no concurrency in the Feature Development Phase which helps accommodate JIT. The Agile RGM supports JIT to a greater extent during the Feature Development Phase than the subsequent phases. Concurrency is discussed in section 3.6.

The following sections describe the phases of the Agile RGM.

3.1 Education Phase

The education phase is the first phase of the Agile RGM model and is designed to help the stakeholders (stakeholders include business analysts, customers, users, developers, project managers and testers) establish rapport. 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. Figure 3.3 shows the education phase extracted from the Agile RGM.

The major objectives of this phase are:

- Establish rapport among the various stakeholders – Effective communication among the stakeholders is essential to gathering the customer and user needs. The customers and users often find it difficult and uncomfortable to talk to the development team about their needs and expectations for a new system. Hence, there is a need to establish rapport among the various stakeholders in order to ensure that the user needs and intents are elicited correctly.

- Learn the business process of the customers – The development team has to learn about the current system and how it satisfies the customer's needs and the existing business process of the customer. This ensures that the development team has a better understanding of the current system and the expected functionality of the proposed system. Ethnographic studies can be carried out to understand the business process of the customers.

- Explain the Agile RGM to the customers and users – The customers and users need to be familiarized with the Agile RGM. This allows the customers and users to understand the workflow through the Agile RGM as they are involved through the complete software development process.

- 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.

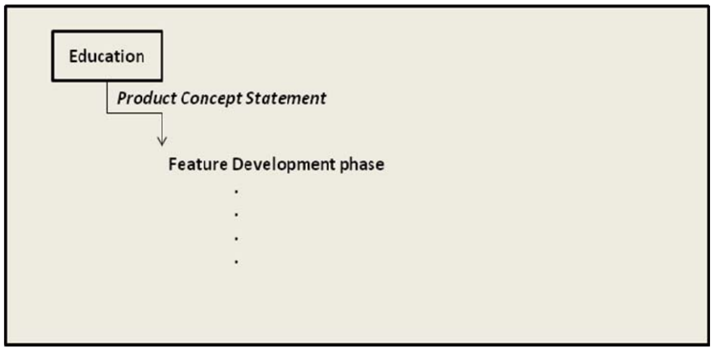


Figure 3.3.Education Phase

The output of this phase is a **Product Concept Statement** [27] which is a brief descriptive summary of the product. It would answer the following questions:

1) Who are the product users?

2) What will the product do?

3) What problem(s) will the product solve?

The Product Concept Statement will help the stakeholders focus on the product development. It can be refined during the various iterations of product development. Currently, distributed teams are very common. Hence, face-to-face communication is not always possible. In these cases, tele- or video-conferencing may be used. The Product Concept Statement serves as the input to the Feature Development Phase. The stakeholders determine the expected functionality of the system from this statement. The next section describes the Feature Development Phase.

3.2 Feature Development Phase

The Feature Development Phase (Figure 3.4) is the second phase of the Agile RGM model where the release level features are identified. Release-level features are defined at the highest level of abstraction. Paraphrased from [18], a feature can be defined as the smallest set of functionality that provides business value to the customer. "Business value is something that delivers profit to the organization paying for the software in the form of an Increase in Revenue, an Avoidance of Costs, or an Improvement in Service (IRACIS) which are discussed below:

- Increase Revenue – Will this functionality increase the revenue generated by the system?
- Avoidance of costs – Will this functionality improve efficiency and reduce wasteful processes?
- Improvement in Service – Will this application help us provide timely service and information to others in a better way?" [19].

The Product Concept Statement created during the education phase serves as the input to the Feature Development Phase as shown in Figure 3.4. The stakeholders identify the expected functionality from the Product Concept Statement. This phase is iterative and the release-level features are identified over a number of iterations.

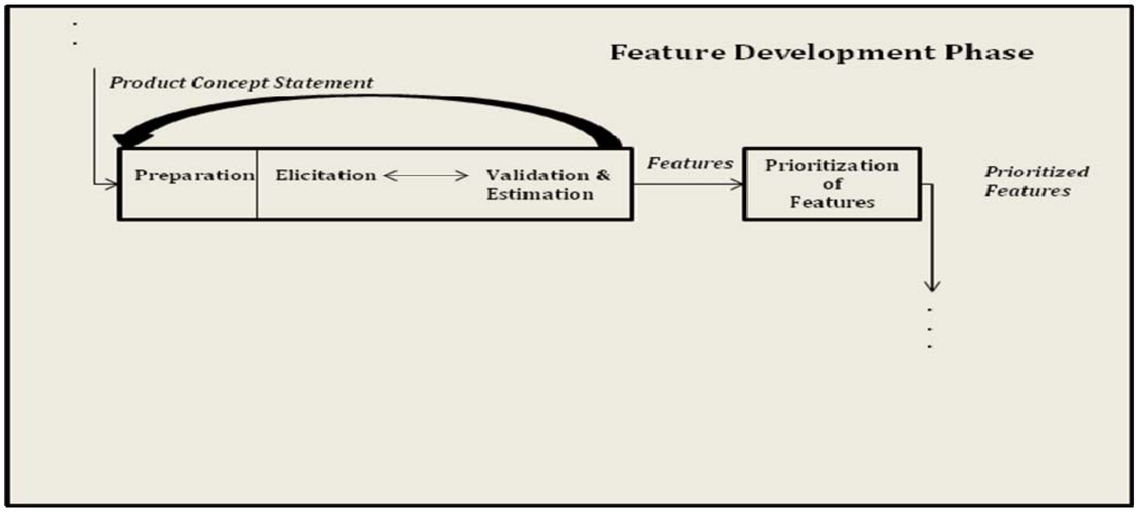

Feature Development Phase
Product Concept Statement
Preparation
Elicitation
Validation & Estimation
Features
Prioritization of Features
Prioritized Features

Figure 3.4.Feature Development Phase

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. Examples for features are given in Table 3.1.

Table.3.1. Examples for features

Examples for features: **Online payment** is a feature that can be identified for an e-commerce company website. **Search catalog** can be another feature for the same project.

The elicitation activity in this phase is a pre-determined meeting called for to identify features. The features are elicited from the customers. The elicited features are then validated. The development team checks the validity of the identified features by discussing the features with the customers. The developers then estimate the time required for the completion of each feature.

As mentioned earlier, the features are identified over a series of iterations. The stakeholders should strive to identify all the features upfront before prioritizing them in order to determine the scope of the system.

The customers prioritize the features based on their business value. These prioritized features are stored in a stack known as the prioritized feature stack. Features are ordered in the stack based on their priorities.

Customers choose features to be implemented during a release cycle from the prioritized feature stack. One feature or a subset of the prioritized features is chosen for development during a release cycle. This subset of prioritized features serves as the input to the Story Development Phase. The major objectives of the Feature Development Phase are:

- Identify release level features – The stakeholders meet to determine the release-level features which provide business value to the customers and users.
- Validate the identified features and estimate the time for completing the features – The stakeholders validate the identified features. The developers estimate the time required to implement each feature.
- Prioritize the set of identified features – The customers prioritize the identified features and store them in a prioritized feature stack.

The various activities of the Feature Development Phase are:

1. **Preparation –** This activity is to arrange for a meeting among all the stakeholders to identify the major features of the system to be built. Also, issues identified during the previous iterations of the Feature Development Phase are resolved. Such meetings can include brainstorming sessions, interviews and focus groups.
2. **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.
3. **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. If they identify

changes to the existing features or recognize the need for new features, another iteration of the feature development phase is carried out.

The stakeholders decide the time frame for a release which could range from 2 to 3 months. Each release cycle comprises of iterations and each iteration lasts a few weeks. Developers estimate the time for completion for each feature. The time estimates for a feature factor in time for gathering details from customers, coding, testing and helping customers plan and automate acceptance tests. If the time estimated for a feature is greater than the time frame for a release cycle, the feature is decomposed during the next iteration of the feature development phase. One or more features can be considered for a release.

4. **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. Some of the techniques used for prioritization are listed below:

   - Use *MoSCoW* (Must have, Should have, Could have, Won't have this time) [20] rule where *must have* features are fundamental to the system, *should have* features are considered mandatory if there are no time constraints, *could have* features are those that can be ignored if time runs out and *won't have this time* features are saved for a later release.

   - Determine which feature sets make up the minimum product of value to the customers [16] and assign priorities accordingly.

   Prioritization of features should take into account strong dependencies among feature sets [16]. Such dependencies are usually domain specific. Sessions with the domain experts would help understand such dependencies and resolve conflicting priorities.

The output of the feature development phase is a **prioritized feature stack**. The feature stack contains the prioritized features ordered by their priorities. The features are recorded on index cards. Hence, the prioritized feature stack is a collection of index cards containing features and sorted by their priorities.

If a release level feature is identified late during the development life cycle, the time required for its completion will be estimated, its priority determined and it will be added to the prioritized feature stack. On identifying new features, existing priorities should be reassessed.

Each feature to be built during the current release is decomposed into stories during the story development phase described in the next section.

3.3 Story Development Phase

The features chosen for development during the current release cycle serve as the input to the Story Development Phase. Each feature is decomposed into stories. "Stories are descriptions of user- or customer-valued functionality" [20]. Stories are defined at a lower level of abstraction when compared to the features. Each identified feature for a current release is decomposed into stories. 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. Examples for stories are listed in table 3.2.

Table.3.2. Examples for stories

Examples for stories:

Consider the online payment feature mentioned in table 1. The following are two stories which can be created for this feature:

1. **A user can pay by credit card**
2. **A user can use an e-check to make a payment**

For the search catalog feature, the following story can be created:

1. **A user can search for a product on various fields**

The Story Development Phase is an independent process carried out for each feature. Each feature is decomposed into a set of stories. If multiple teams are involved, more than one feature can be decomposed into stories in parallel by adopting concurrent Story Development Phases. Concurrency in the Agile RGM is discussed in section 3.7.

The Story Development Phase and its activities are shown in Figure 3.5. Each feature to be implemented during the current release cycle is decomposed into stories. The development team elicits stories from the customers and users during pre-determined meetings. The elicited stories are validated by the customers. The developers estimate the time required for each story to be completed. The validated stories are then prioritized by the developers. The prioritization is based on dependencies among the stories. These prioritized stories are stored in a prioritized story stack. This prioritized story stack can take the form of a bunch of index cards or electronic cards with stories and their priorities recorded on them. The stories are elicited over a number of iterations.

When an initial set of prioritized stories is created, the team can proceed to the Task Development Phase and create task lists for the stories. If other stories are identified further along the development process, they are validated, estimated and prioritized relative to the already existing stories and stored in the story stack.

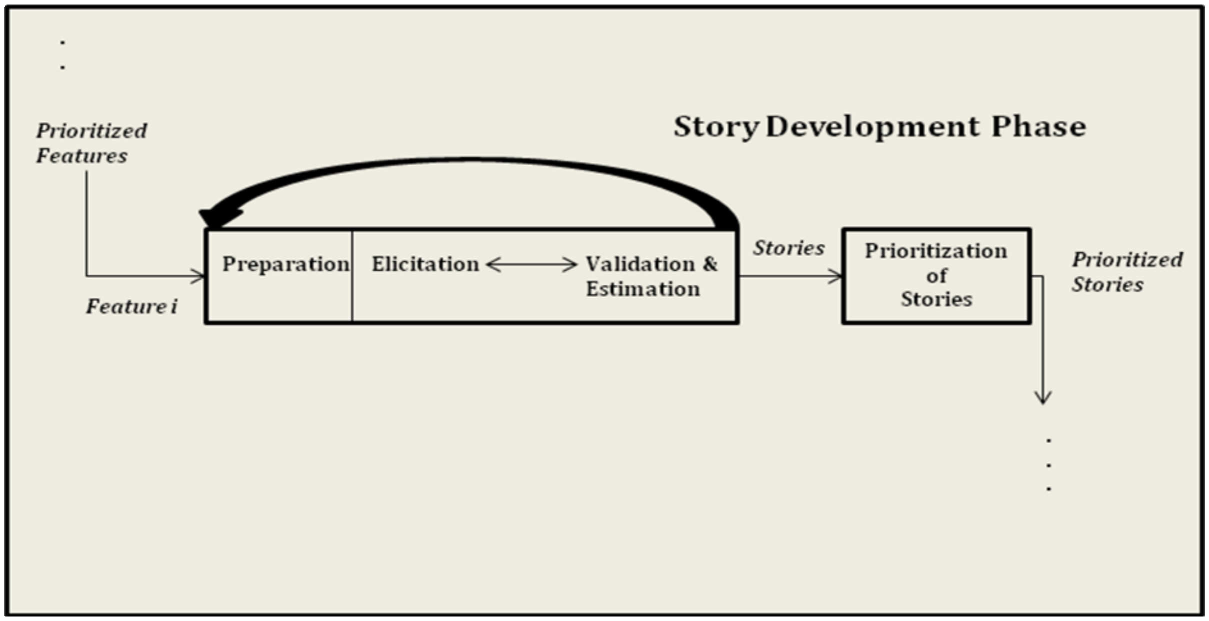


Figure 3.5.Story Development Phase

The objectives of the Story Development Phase are:

- Create user stories – The stakeholders meet to create stories for each feature to be implemented during the current release cycle. The stories are elicited over a number of iterations.
- Validate the stories and estimate the time for completing a story – The stakeholders validate the identified stories over the many iterations and the developers estimate the time required to complete the stories.
- Prioritize the stories – The developers prioritize the stories and store them in a prioritized story stack.

The various activities of this phase are:

1. **Preparation** – This objective of this activity is to arrange for a meeting among the stakeholders is to create stories for a subset of prioritized features.
2. **Elicitation** – The chosen feature is decomposed into stories. This process takes into account the agile practice of no BRUF and the details are gathered on a just-in-time basis. Stories are created for all the user classes identified during the education phase. Common techniques for eliciting stories include *interviews, observing users interact with software, questionnaires and story-writing workshops* [20]. A story writing workshop is a brainstorming session involving all the stakeholders dedicated to creating stories. Customers write the acceptance criteria for each story. Agile methods focus on delivering functionality of maximum business value to the customers. Customer and user preferences take precedence. Hence, the customers write the acceptance criteria and later validate the developed code against these criteria.
3. **Validation and Estimation** – The stories created are validated and the developers estimate the time required for completing each story. Validation involves discussing the stories with the customers, creating screen designs and paper prototypes. If there are changes to the identified stories or if need for new stories is recognized, another iteration of the story development phase is carried out.

As mentioned earlier, the product development cycle spans multiple releases and each release is divided into iterations. The time for each story is usually estimated in *story points* which are relative estimates of complexity, effort or duration of a story [20]. Each agile team has its own definition for story points. Story points can be defined in terms of number of hours, days or weeks of work.

4. **Prioritization –** The developers determine the dependencies if any among the stories and prioritize the stories accordingly. They also consider the customer and user preferences [20]. Prioritization techniques discussed in the previous section can be used. 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 phase is a **prioritized story stack**. The stories are identified over a series of meetings. However, after a story or an initial set of stories are identified, tasks can be created and developed.

Creating stories for features on a just-in-time basis greatly aids accommodating change even late in the development cycle. Changes to stories identified late in the development lifecycle are logged and are addressed in the current or the next iteration. If new stories are identified, priorities are recomputed and the new stories are also stored in a story stack.

The stakeholders select a subset of the prioritized stories based on their priority and time estimates to be developed during the current iteration. Each iteration lasts for about 2 to 4 weeks. This timeframe is decided by the development team. This subset of stories to be implemented during the current iteration is fed as input to the Task Development Phase described in the next section.

3.4 Task Development Phase

The stories chosen to be developed during the current iteration serve as the input to the Task Development Phase. Each story is decomposed into tasks by the development team. The Task Development Phase is an independent process carried out for each story. 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 [20]:

- 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 for each story are created during one or multiple iterations of the Task Development Phase by the developers. These 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. Concurrency in the Task Development Phase is discussed in section 3.7. Examples for tasks are given in table 3.3.

Table 3.3.Examples for tasks

Examples for tasks

Consider the story “A user can pay by credit card” described in table 2. The following are some of the tasks for this story:

1. **Code credit card details page**
2. **Code order details page**
3. **Code credit card authorization page**
4. **Ensure that only Visa and Master cards are accepted**
5. **Code order confirmation page**

Consider the story “A user can search for a product on various fields” given in table 2. The following are some of the tasks for this story [20]:

1. **Code basic and advanced search pages**
2. **Code display results page**
3. **Tweak the SQL query for advanced search**

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. The developers then estimate the time required to complete each task. Each task requires no more than a few hours to be developed as opposed to each story which takes a few days to be implemented. Hence, estimation activity is performed even at the task level. The Task Development Phase and its activities are shown in Figure 3.6 below. The task lists are created over a number of iterations. As mentioned earlier, task lists for more than one story can be created in parallel.

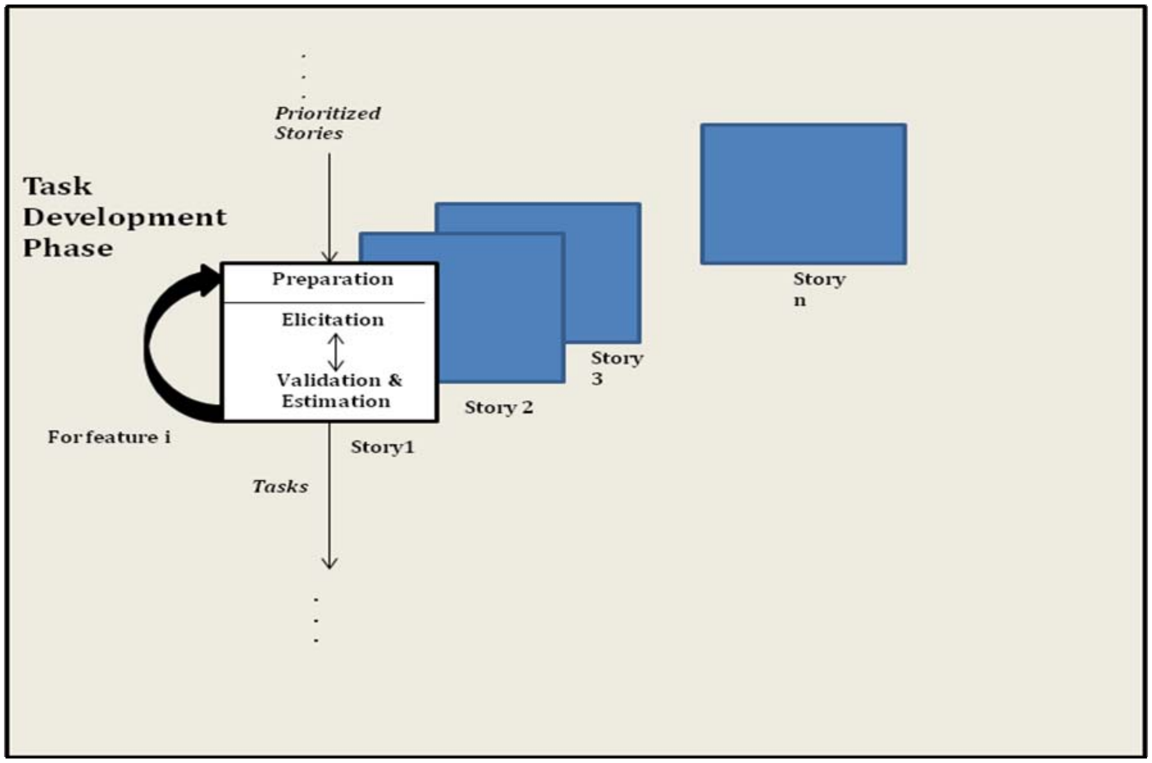


Figure 3.6.Task Development Phase

The objectives of the Task development Phase are:

- Create tasks for each story – Each story is decomposed into tasks. These tasks are essentially to-do lists for the developers to guide them through the development process. The development team is responsible for creating the task lists.

- Validate the tasks and estimate the time for completing the tasks – The development team performs an informal review of the tasks to ensure that they have not overlooked any detail. The developers estimate the time required to complete the tasks.

The various activities of the Task Development Phase are:

1. **Preparation –** This activity is to arrange for a development team meeting to create a task list for a story. Issues identified in the previous iterations may be resolved.

2. **Elicitation –** The development team thinks out loud and creates the task lists. The lists of tasks are usually recorded on white boards. Each developer volunteers to take responsibility for one or more tasks.

3. **Validation and Estimation –** The development team does an informal review of the task lists and missing details are filled in just-in-time. Each task is then estimated to take no more than a few hours to be implemented.

   There are no rules regarding the size for a task [20]. Each story is broken into tasks in order to enable more than one developer to work on different parts of a story. Usually, each task is estimated to take a few hours. If more than one developer can work on a task, the task is further decomposed in order to facilitate concurrent development. Section 3.6 discusses concurrency in the Agile RGM.

The tasks can be developed as and when they are created. The output of this phase is a **task list** created for each story. These task lists serve as the input to the Development Phase which is discussed in the next section.

3.5 Development Phase

The focus of the Agile RGM is to provide a soft-structured approach to gathering requirements in the form of features, stories and tasks. However for completeness, we have included a component for implementation of these requirements which is the Development Phase. Also, the model has been designed for small scale, non- mission and life critical systems. This assumption requires viewing the model not only from a requirements engineering perspective but also from a development perspective. Hence, describing the development process is justified. The tasks created during the previous Task Development 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 Agile RGM reflects the agile philosophy and hence, we suggest using TDD and Customer Acceptance Testing as activities during this phase.

Each task created earlier is developed in 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. Figure 3.7 shows the Development Phase. More than one developer may work on the tasks created for each story. Each developer chooses a set of tasks for the story to be implemented based on their skills. After completion, each task is integrated into an existing code base. As mentioned earlier, development and testing of multiple tasks can be completed concurrently. Each iteration yields workable software.

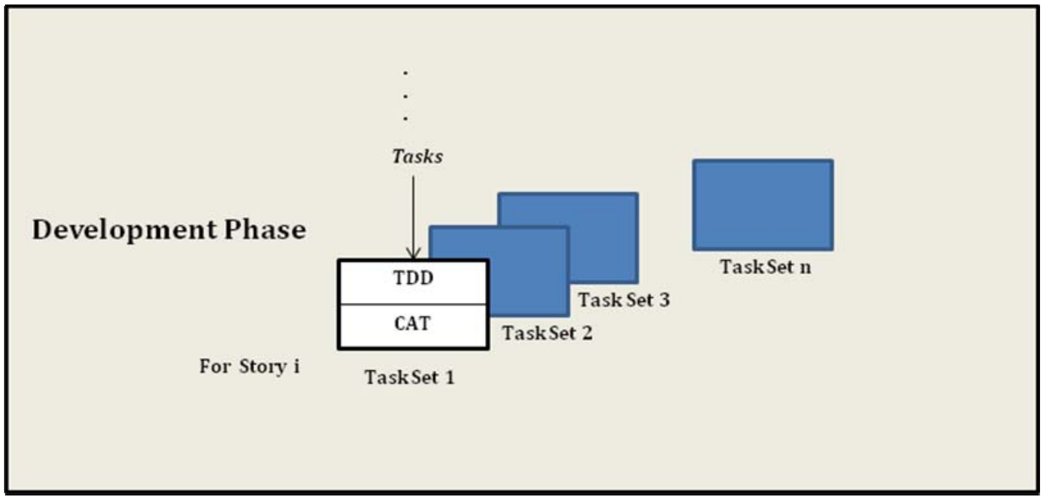


Figure 3.7.Development Phase

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 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 Development Phase are:

1. **Test Driven Development** (TDD) – "Clean code that works" is the goal of TDD. The steps involved in using TDD [22] are given below:

    - Write a test that fails
    - Write code that would make the test work
    - Refactor the code to improve its structure and eliminate dead code

    "Developers follow short cycles of test-code-test-code and so on" [22]. The rule is to not write operational code until a test fails. The suite of tests created ensures that each function has its own set of tests. TDD produces well factored, testable and maintainable code. The code created is then tested against the acceptance criteria defined by the customers and users.

2. **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. Developers create additional tests which augment those written by the customers. It is possible that code produced during a previous iteration be broken during subsequent iterations. Hence, acceptance tests from all the

previous iterations should be run in order to ensure that all pieces of code show expected behavior.

Executing acceptance tests can be time consuming. Hence, teams should try to automate the acceptance tests. *Framework for Integrated Test* (FIT) [23] and FITNesse [24] (uses FIT) are tools available for automating acceptance tests [20]. These tools enable the customers, developers and testers to learn about the expected and observed functionality of the system under development. The customers specify the acceptance criteria and the developers write tests based on these criteria. These tools automatically check developed software for compliance with the acceptance criteria. Both FIT and FITNesse provide tabular formats for writing tests.

The code after acceptance testing is integrated into the existing code base.

3.6 Mapping between Agile RGM and RGM

In this research we have used the RGM to structure the Agile RE process. The Agile RGM preserves the structure of the RGM with minimal impact on agility. The activities and the objectives of the RGM are mirrored in the Agile RGM. This section discusses the mapping between the Agile RGM and the RGM. Table 3.4 shows the mapping between the Agile RGM and the RGM.

The Big Picture

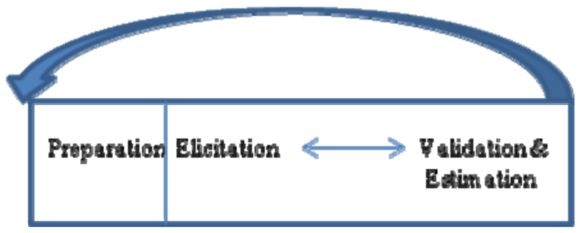
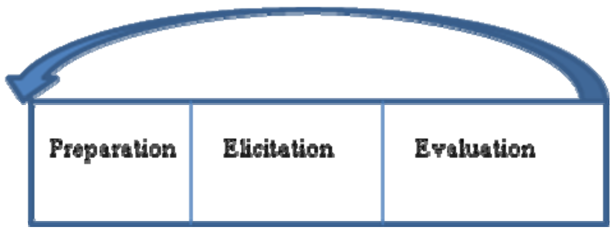
The RGM is a structured approach to gathering requirements. It is an iterative process. The RGM facilitates identifying requirements over multiple iterations and refining of the identified requirements. Formal requirements are produced during the process. The major activities of the RGM are Indoctrination, Preparation, Elicitation and Evaluation which were discussed previously in Section 2.5.

The Agile RE process suggests defining requirements at a high level which are commonly called features or stories. Further details for each feature or story to be implemented during the current iteration are gathered in a JIT fashion. The Agile RGM reflects this principle. Initially, release level features are produced during the Feature Development Phase and each feature is further decomposed into stories during the Story Development Phase. Each story identified during the

Story Development Phase is then decomposed into tasks during the Task Development Phase. In the Agile RGM, the notion of requirements is reflected in the form of features, stories and tasks. The Agile RGM applies the RGM thrice for gathering features, stories and tasks during the Feature, Story and Task Development Phases respectively. Each of these three phases show an iterative process that mirror the RGM.

The rationale behind the Education phase and the activities of Preparation, Elicitation, Validation and Estimation of the Feature, Story and Task Development Phases of the Agile RGM is similar to that of the Indoctrination, Preparation, Elicitation and Evaluation activities of the RGM.

Table 3.4.Mapping between Agile RGM and RGM

Agile RGM	**RGM**
 Structure of the Iterative component of Feature, Story and Task Development Phases of the Agile RGM	 Structure of the Iterative component of the RGM
Education: 1. Establish rapport among stakeholders 2. Developers learn the business process of the customers 3. Create a Product Concept Statement	Indoctrination: 1. Developers learn the business process of the customers 2. Customers are educated about the RGM
Preparation 1. Arrange for sessions to elicit features, stories and tasks during Feature, Story and Task Development Phases respectively	Preparation 1. Arrange for a meeting to elicit requirements
Elicitation 1. Elicit features, stories and tasks during Feature, Story and Task Development Phases	Elicitation 1. Elicit requirements
Validation and Estimation 1. Validate features, stories and tasks 2. Estimate time required to implement the identified features, stories and tasks	Evaluation 1. Review identified requirements 2. Record issues if any

Education and Indoctrination

The Education Phase of the Agile RGM and the Indoctrination Phase of the RGM have similar objectives. Both serve as the education component for the customers and the development team. During the Education Phase of the Agile RGM, a high level mission statement called the Product Concept Statement is also produced. Such a statement is not produced as a part of the Indoctrination phase of the RGM.

Preparation

The Preparation phase of the RGM is mirrored in the Feature, Story and Task Development Phases of the Agile RGM. The objectives of this activity remain the same in the Agile RGM and the RGM. The main objective of this activity is to arrange for a meeting among the stakeholders in order to elicit requirements when adopting the RGM and features, stories and tasks when adopting the Agile RGM.

Elicitation

The objective of this activity in the RGM is to gather requirements from the customers and the users. Elicitation in the Agile RGM deals with gathering features, stories and tasks during the Feature, Story and Task Development Phases respectively. Both models suggest open-ended interviews, brainstorming sessions, focus groups, etc., to elicit requirements, features, stories and tasks. In the RGM, the Elicitation activity has to be complete before proceeding to Evaluation. The requirements have to be gathered before reviewing. In the Agile RGM, the elicitation of features, stories and tasks can be done in parallel to validation due to constant customer collaboration.

Validation and Estimation, and Evaluation

The Evaluation activity of the RGM focuses on reviewing the identified requirements and recording issues if any. This is very similar to the Validation activity of in the Agile RGM. Validation is a part of the Feature, Story and Task Development Phases of the Agile RGM. The identified features, stories and tasks are validated. In the Agile RGM, after the features, stories and tasks are validated, the time required for their implementation is also determined. Estimation of time required for completion of requirements is not a part of the RGM.

3.7 Concurrency in the Agile RGM

As mentioned earlier, each feature is decomposed into stories and each story into tasks. Hence, the Story Development and Task Development phases are independent processes carried out in order to decompose one feature into stories and one story into tasks respectively. However, if multiple teams are involved, concurrent Story and Task Development efforts are feasible. This section discusses the support for concurrency in the Agile RGM.

The Agile RGM supports concurrency from the Story Development Phase onwards. Concurrent efforts are supported if multiple teams are involved. As the Story Development, Task Development and Development Phases are independent processes, parallel efforts to decompose more than one feature into stories and more than one story into tasks is possible.

The Feature Development Phase does not support concurrency. The stakeholders should strive to identify all the features upfront in order to determine the scope of the system. However, if new features are identified late in the development process, priorities for the new features are assigned relative to the already identified features and stored in the prioritized features stack. Only one feature or a subset of features is decomposed into stories during a release cycle. This just-in-time gathering of details helps accommodate change.

As mentioned earlier, the Story Development, Task Development and Development Phases support concurrency. In the case of the Story Development Phase, multiple features can be decomposed into stories by adopting parallel Story Development efforts. *Each Story Development effort produces stories for one feature.*

Similarly, concurrent Task Development efforts are feasible when multiple teams are involved. Hence, more than one story can be decomposed into tasks by adopting concurrent Task Development efforts. Each story of one feature can be decomposed into tasks in parallel. . Multiple tasks can be developed and tested concurrently. Hence, concurrent development efforts are supported by the Agile RGM.

The advantage of concurrency is that the development of the product progresses faster. As multiple teams are involved and simultaneous efforts to develop the tasks are undertaken, rapid development is possible. However, concurrent development efforts reduce JIT gathering of details. The stakeholders cannot wait until the latest responsible moment to gather details. Hence, changes identified to stories and tasks may affect already created stories and tasks. Supporting concurrency limits the advantages of the JIT philosophy.

There is no support for concurrency during the Education and Feature Development Phases. During the Feature Development Phase, the stakeholders strive to identify all the features upfront in order to determine the scope of the system. Hence, there is no support for concurrency during the Feature Development Phase. In the absence of support for concurrency, JIT can be employed to a greater extent and changes to requirements can be accommodated even late in the development lifecycle.

Concurrency is supported in the Story Development, Task Development and Development Phases. This enables faster development of the product. However, the extent to which JIT can be used is reduced.

The Agile RGM supports both high and low levels of concurrency thus taking advantage of both rapid development and JIT philosophy. The development team makes decisions regarding the degree of concurrency to be adopted when following the Agile RGM process.

3.8 Summary

The Agile RGM has been designed for short term (less than one year development period, 60 to 90 day release cycle), non-mission and life critical agile projects. As the time frame for the project development is also considered, the Agile RGM describes the development process as well. It is an iterative and incremental model which focuses on just-in-time gathering of details, evolutionary requirements, no BRUF, refactoring, direct stakeholder involvement and minimal documentation. The Agile RGM can accommodate change even late in the development lifecycle. Agile RGM is designed to fit in with existing agile development methods as described in the next chapter (Chapter 4).

4. Substantiation of the imputed effectiveness of the Agile RGM

The Agile RGM is an independent model designed to work with any of the existing agile methods. The model reflects the focal values of the agile philosophy and the principles stated in the agile manifesto which would help in integrating it with the existing agile methods. Agile RGM serves as a guideline to the agile approach to development with a greater focus on RE and suggests techniques that practitioners can adopt at each phase of the development process. The objective of this chapter is to substantiate the claim that the model is independent and can work with any of the existing agile methods. In order to substantiate that claim, it should be established that the Agile RGM reflects the agile philosophy as described in the agile manifesto. This chapter first discusses how the Agile RGM reflects the agile philosophy, and the principles and practices of the agile approaches to RE and software development. It also describes how the Agile RGM can be integrated with Scrum, XP and Feature Driven Development.

4.1 Agile RGM – Agile Manifesto

The Agile RGM can be used with the existing agile methods. This is feasible as the Agile RGM reflects the values and principles stated in the Agile Manifesto [6]. This section discusses how the Agile RGM is reflective of the values and principles stated in the Agile Manifesto.

The Agile RGM reflects the focal values and principles stated in the agile manifesto. The following paragraphs in this section discuss the focal values and principles of the Agile Manifesto and how the Agile RGM 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 each phase of the Agile RGM in the following ways:

- The Agile RGM emphasizes direct customer involvement. Customers and users are involved throughout the process.
- The activities described in the Education Phase, Feature, Story and Task Development Phases 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 task development phase when the stories are decomposed into to-do lists for the developers.

"**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 Agile RGM emphasizes on delivering working software to the customers at the end of each iteration rather than on documentation. Documentation produced during the phases is minimal and mostly consists of features and stories recorded on index cards. At the end of the development phase, the new code is integrated with the existing code base. The working software produced at the end of each iteration is potentially shippable to the customers.

The Agile Manifesto states that the customers and the development team should be prepared to modify plans in order to accommodate change even late in the development lifecycle. This is conveyed by the fourth focal value "**Responding to change** over following a plan". The Agile RGM adopts the just-in-time philosophy to gather user needs, decompose features into stories and stories into tasks. The developers estimate the time required to complete the features, stories and tasks just-in-time. This allows the team to respond to changes in user needs and to adjust the amount of work that they wish to accomplish during a given time frame. The model encourages the stakeholders to accommodate change even late in the development process. This is reflective of the above mentioned focal value.

The Agile RGM reflects the principles stated in the agile manifesto and is discussed in this paragraph. The model emphasizes on direct customer involvement throughout the development process. Each phase in the model requires close interaction between the development team and the customers. The Task Development Phase in particular, encourages face-to-face communication among the development team. Agile RGM adopts just-in-time philosophy to gather details at every phase which helps accommodate change even late in the development process. The model is iterative and incremental and hence makes continuous delivery of working software that is of value to the customer feasible.

The agile philosophy discourages formalism and focuses on lightweight methods. We deviate a little from this principle in designing the Agile RGM which adds some structure to the agile approach to RE and spans the complete development process. However, the Agile RGM is a soft-structured process which is intended to serve as a guideline to practitioners. The model provides supporting structure to the agile approach to RE. Since the development time of the system is considered, the model describes the complete development process.

Some of the agile principles stated in the manifesto which dictate the working conditions to be provided and discuss the self-organizing nature of the teams are not reflected in the model. These principles are more related to the development environment rather than the development process which is the focus of the Agile RGM. For example, “At regular intervals, the team reflects on how to become effective, then tunes and adjusts its behavior accordingly” [6] is one of the principles stated in the manifesto. The Agile RGM does not explicitly reflect this principle. However, if the Agile RGM is used in practice, it can fit within the development environment and the principles stated in the manifesto will be reflected.

4.2 Agile RGM – Agile RE

The Agile RGM attempts to add some structure the agile approach to RE. It reflects the principles and practices of the agile approach to RE discussed previously. The model adopts the practices of evolutionary requirements, JIT, direct customer involvement, etc. and are discussed below:

1. Evolutionary Requirements – This agile practice states that requirements should evolve over time. In the Agile RGM, the stakeholders identify features initially to determine the scope of the system. Only a subset of the identified features is decomposed into stories and the stories are in turn decomposed into tasks. Hence, the requirements are not

identified upfront. The requirements evolve over time. The Agile RGM adopts the practice of evolutionary requirements.

2. Just-In-Time – The JIT philosophy adopted by agile RE states that the details of features/ requirements should be gathered in a JIT fashion and not upfront. The Agile RGM adopts the JIT philosophy at every phase to gather details from the customers. Also, JIT helps the team adapt to changes to requirements and the amount of work to be done within a pre-determined time frame.

3. Minimal Documentation – Agile RE focuses on minimal documentation. The documentation created is usually informal and consists of a set of index cards containing features. The model suggests creating minimal documentation in the form of features, stories and tasks recorded on index cards.

4. Verification and Validation – The agile approach to RE does not explicitly specify V&V efforts. Also, verification is more of a validation process. As the customers are available almost at all times during development, validation is done implicitly and just-in-time. However, the Agile RGM explicitly states a validation activity in each phase to ensure that the features, stories and tasks reflect the needs and intents of the customers and users. The objective in defining a validation activity is to ensure that no detail is overlooked and all the needs and intents of the customers are captured.

5. The agile approach to RE emphasizes on fixing the time frame and resources before the features are estimated. In the Feature Development Phase of the Agile RGM, the developers are aware of the time frame for the releases before they estimate the features. If a feature is estimated to take longer than the time frame for a release cycle, the developers decompose the feature in order to accommodate the timeline for the project.

6. Direct customer involvement – Agile RE is largely dependent on active customer involvement in the development lifecycle. The Agile RGM activities encourage close interaction among all the stakeholders throughout the process in order to elicit needs, gather details just-in-time and validate the output of each phase.

7. Agile RE emphasizes iterative and incremental implementation of requirements. Each phase in the Agile RGM is iterative and the system is developed in increments. Requirements are allowed to evolve over time which facilitates incremental development.

8. Agile methods treat requirements like a prioritized stack. The Agile RGM preserves this idea. Features, stories and tasks are created instead of a formal requirements specification document. The output of the Feature and Story Development Phases are prioritized feature and story stacks respectively. The prioritization is done based on the value of a feature or story to the customer.

4.3 Agile RGM – Agile Development

Agile practices such as refactoring and TDD are adopted for implementing features, stories or tasks. The Agile RGM adopts these practices which are common to the agile approach to software development. Some of the agile development practices adopted by the Agile RGM are discussed below:

1. Test-Driven Development – Developers write tests first before writing code. The rule is to not write operational code until a test fails. Agile RGM suggests using TDD in the Development Phase to implement tasks. TDD is considered suitable for small scale system development. Since the Agile RGM is designed for systems with development periods of less than 1 year, it suggests using TDD for implementing the tasks during the development phase.

2. Refactoring – Refactoring is the process of altering the structure of code while explicitly preserving its behavior. Refactoring is a part of TDD. Since the Agile RGM suggests using TDD for development of tasks, code refactoring is also adopted.

3. Customer Acceptance Tests – Acceptance tests ensure that the system developed meets the expectations of the customer. During the Story Development Phase of the Agile RGM, the customers specify acceptance criteria for each story. The customers later test the working software produced at the end of each iteration against the criteria specified previously. The Agile RGM suggests the use of tools such as FIT and FITNesse to automate the acceptance tests.

4. JIT – Adopting the just-in-time philosophy is a core agile practice. Practitioners defer making decisions or gathering details from customers until the latest responsible moment. The Agile RGM emphasizes on JIT in all phases as mentioned previously.

5. Incremental development – Agile methods emphasize on incremental development and the Agile RGM 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 only a subset of the features identified during the Feature Development Phase is chosen for development during a release cycle.

4.4 Agile RGM – Agile Methods

The Agile RGM is designed as an independent model that can be used in practice with any of the existing agile methods like Scrum, XP and Feature Driven Development. As the model reflects the agile philosophy, it can be integrated with any of the existing agile methods. This section describes how the Agile RGM can be integrated with some of the agile methods namely Scrum, XP and Feature Driven Development.

4.4.1 Agile RGM – Scrum

This section discusses the similarities among Scrum [17] and the Agile RGM and describes how the Agile RGM can be integrated with the Scrum process.

Scrum is an agile approach developed to manage the software development process. It attempts to structure the development process by employing an iterative and incremental process skeleton that includes a set of pre-determined processes and roles. However, it does not specify implementation techniques. Hence, practitioners adopting Scrum also use XP or other agile methods that can provide implementation techniques. Unlike Scrum, the Agile RGM specifies activities and suggests techniques that can be used. Scrum and the Agile RGM have many similarities. The following paragraphs discuss the similarities among Scrum and Agile RGM.

Incremental Development

Scrum dictates that the product be built in increments. Work is structured in iterations called sprints lasting 2 - 4 weeks. At the end of each sprint, a potentially shippable product increment is produced. Like Scrum, the Agile RGM is an iterative and incremental model which facilitates developing software in increments and adds structure to the agile approach to software development. An iteration which lasts 2 – 4 weeks is comparable to a sprint in Scrum. At the end of an iteration, working software of value to the customer is produced.

The JIT philosophy and iterative and incremental development approach adopted by both Agile RGM and Scrum help accommodate changing requirements even late in the development lifecycle.

Scrum - *Initial Meeting*

Each team that adopts Scrum usually convenes an initial meeting among the stakeholders called *Scrum 0*. The objective of this meeting is for the development team to learn the business process of the customers. Scrum 0 is very similar to the Education phase of the Agile RGM as both have similar objectives.

Scrum Ceremonies

The three ceremonies in Scrum are the Sprint *Planning meeting, Daily Scrum meeting and Sprint Review meeting* which were discussed previously. During the Sprint Planning meeting, a *product backlog* is created which consists of a set of prioritized features identified by the stakeholders. A subset of prioritized features from the product backlog is chosen for development and these features are broken down into *sprint tasks* which are the development activities required to implement a feature. The sprint tasks are stored in a *sprint backlog*. The *Sprint Planning meeting aligns closely with the Story Development and Task Development Phases of the Agile RGM*. The output of these phases, the prioritized story stacks and task lists are comparable to the product and sprint backlog. However, the Agile RGM includes a higher level of abstraction in the form of features. Also, the Feature Development Phase is associated with planning at a release level where each release cycle lasts for about 60 – 90 days. The release cycle is similar to a Scrum cycle which consists of sprints.

The *Sprint Review Meeting* is arranged at the end of each sprint in order to demonstrate the potentially shippable product and allow the customer to validate the output. This is similar to the

Customer Acceptance Testing activity which is a part of the Development Phase in the Agile RGM.

Integration of Agile RGM with Scrum

The similarities among Agile RGM and Scrum are described in the previous paragraphs. These similarities encourage the integration of the two approaches. As mentioned earlier, Scrum provides a process skeleton only. There is no specification of activities that are to be carried out during a scrum cycle and no suggestions for techniques that can be used. The Agile RGM on the other hand, addresses the above issues as it provides delineation of activities for each phase and suggests techniques that can be used. Hence, using the Agile RGM with Scrum would prove helpful to practitioners.

The Education and Feature Development Phases can be made a part of Scrum 0 as both these phases can be considered as upfront activities. At the end of Scrum 0, a subset of features to be developed during the next scrum cycle is identified. This subset of features is extracted from the prioritized feature stack. The next scrum cycle can be treated as a release cycle and consists of a number of sprints or iterations. During a sprint, activities described in the Story Development, Task Development and Development Phases of the Agile RGM are carried out. During the subsequent scrum cycles, subsets of features from the prioritized feature stack are passed through the Story Development, Task Development and Development Phases over many sprints that make up these cycles. At the end of each sprint, a potentially shippable product increment is produced.

Using the Agile RGM within a scrum process can serve as a guideline to the complete development lifecycle. The practitioners will be made aware of the activities to be carried out during each scrum cycle and the practices that can be used. The artifacts suggested by the Scrum process are comparable to those of the Agile RGM. Hence, no additional effort is required to create them.

As mentioned in the previous paragraphs, Agile RGM and Scrum have many similarities. Agile RGM can be integrated with Scrum and adopted by practitioners.

4.4.2 Agile RGM – XP

This section discusses the practices common to XP [14] and the Agile RGM. eXtreme Programming (XP) is a set of practices that is ideal for small to medium teams which have to develop software quickly in the face of rapidly changing requirements. XP practices were discussed in detail earlier. There is no structure to XP. Hence, agile teams usually use XP and Scrum together with Scrum providing some structure to process. The Agile RGM adopts many of the XP practices and can be used with XP instead of Scrum in order to provide some structure to the development process.

Planning Game

The *planning game* in XP is associated with identifying features to be developed during a release cycle and the work to be completed during the next iteration. This is similar to the Feature Development and Story Development Phases of the Agile RGM. The Agile RGM describes the activities to be carried out during these phases and the techniques that can be used.

Refactoring, TDD and Pair-Programming

Refactoring, TDD and *Pair-Programming* are very important XP Practices which are followed in the Development Phase of the Agile RGM and are discussed briefly below.

1. Test Driven Development – Programmers write tests first and then develop code only when the tests fail.
2. Refactoring – This practice is used to restructure the system in order to improve the design and code. Refactoring is achieved by removing duplication, increasing cohesion and lowering coupling of code.
3. Pair Programming – Two programmers work together at one machine writing code and performing code review simultaneously.

These practices are considered suitable for developing small scale systems. The Agile RGM has been designed for small scale systems with less than one year of development period and hence these practices are suggested for the Development Phase. Both XP and Agile RGM insist on *continuous integration* of new code developed into the existing code base.

Direct customer involvement

XP encourages an *on-site customer* who is available at all times at the developer site to determine requirements, set priorities and answer questions when required. Though the Agile RGM does not emphasize on an on-site customer, it encourages direct customer involvement. Each phase in the model requires close interaction between the customers and the development team.

Acceptance Testing

Customer Acceptance Testing is a practice suggested by both XP and Agile RGM. Customers create acceptance criteria for each feature or story and the system is validated against the criteria specified previously.

Many XP practices are suggested by the Agile RGM. The Agile RGM attempts to add some formalism to the entire development process. Also, as discussed in the previous section, Scrum and the Agile RGM are similar. Hence, teams following XP can use Agile RGM instead of Scrum to structure their process.

4.4.3 Agile RGM - Feature Driven Development (FDD)

This section discusses the similarities among the Agile RGM and FDD [16] and how the Agile RGM can be integrated with FDD. FDD consists of five sequential processes (*develop an overall model, build a features list, plan by feature, design by feature, build by feature*) which focus on the design and coding phases of the development process. It provides the methods, techniques and guidelines to the stakeholders to deliver the system [16]. FDD was discussed in detail in Chapter 2. Documented requirements in the form of formal requirements specification or use cases are available as input to the FDD process. These requirements help identify the scope of the system. The stakeholders then identify features of value to the customers from the documented requirements. The features are defined at a lower level of abstraction and the condition is that *each feature should take no more than 2 weeks to build*. These features are prioritized and project schedules are defined. A set of features to be developed over the next few days (no longer than 2 weeks) is identified. These features are designed and built iteratively. Software is developed in increments.

The major difference between the two approaches is that FDD focuses only on the design and coding phases whereas the Agile RGM spans the entire development lifecycle. Also, the Agile RGM defines features at the release level where each release lasts for 60 – 90 days. A collection of stories that can be completed in 2 weeks is comparable to one feature or a set of features in FDD.

Iterative and Incremental Development

Like FDD, the Agile RGM supports iterative and incremental development. The high level scope of the system is determined by identifying release-level features. These features are then decomposed into stories and stories into tasks prior to the Development Phase. Both Agile RGM and FDD suggest design and development of software on a feature-by-feature basis. However, if multiple teams are involved, more than 1 feature can be developed simultaneously.

Integration of the Agile RGM with FDD

As mentioned earlier, FDD focuses only on the design and coding phases and provides methods and techniques for these two phases. There is no specification of activities for initial steps regarding identifying features and creating prioritized feature lists. The activities and techniques of the initial phases of the Agile RGM namely the Feature Development and Story Development Phases can adopted. Hence, these two phases would span the first 3 steps of the FDD process. The next paragraph describes how the Agile RGM can be integrated with FDD.

Documented requirements serve as the input to the process which can help identify the scope of the system. The stakeholders convert these requirements to features. The Feature Development Phase can be used to create the features that are customer valued functionality from the requirements. It can be mandated that each feature should take no longer than two weeks for development. Hence, features are not identified at the release-level. These features can be prioritized and stored in a prioritized story stack. The features to be built over the next two weeks are identified and these features can serve as the input to the Story Development Phase. Each feature is decomposed into stories. The set of stories for each feature is the input to the Design by Feature and Build by Feature phases of FDD. As the size of the features decrease, the stories also shrink and can be comparable to tasks. Hence, the Task Development Phase can be eliminated.

The activities of the Feature Development and Story Development Phases of the Agile RGM can be adopted for the first 3 steps of the FDD process. This would help structure the RE process. However, if there are no documented requirements available as input to the FDD process, the practices of the first 4 phases (Education, Feature Development, Story Development and Task Development) can be adopted for the initial 3 steps of the FDD process. In this case, it should be mandated that each story should take no longer than 2 weeks to develop. The task lists for each story can serve as the input to the design and coding phases of the process.

The Agile RGM spans the complete development process whereas the FDD focuses only on the design and coding phases of the software development lifecycle. The Agile RGM can be integrated with FDD in order to produce a hybrid approach that would provide the advantages of both FDD and Agile RGM.

4.5 Summary

This chapter substantiates the Agile RGM as an independent model by discussing how it can be used with other agile methods. The Agile RGM is an independent model that reflects the agile philosophy like the existing agile methods. It also adopts the practices unique to the agile approaches to RE and software development like evolutionary requirements, JIT, TDD, refactoring, minimal documentation and iterative and incremental development. Since the model is independent, it can be integrated with the existing agile methods like Scrum, XP and FDD as discussed in this chapter. With XP, the Agile RGM can be used to add some structure to the process. Scrum does not suggest techniques that can be used during the development process which can be remedied by integrating it with the Agile RGM. FDD does not focus on the RE process. This can be addressed by adopting the activities and techniques suggested in the Feature Development and Story Development Phases of the Agile RGM.

5. Generalized Agile RGM (an alternative structure that accommodates Large Scale Systems): An Overview

The objective of this chapter is to provide an overview of an alternative structure to the Agile RGM that can be used for building large scale systems (development period of more than two years). The Agile RGM is a soft-structured approach designed for small scale systems (development period of one year or less) with 60 to 90 day release cycle. As the development period is considered, the model spans the complete development lifecycle from requirements gathering through customer acceptance. The Agile RGM describes an iterative and incremental process and can effectively accommodate changing requirements even late in the development process. Though designed for small scale systems, the Agile RGM can be adapted to large scale systems development. However, using agile methods for large scale systems development raises some issues like the suitability of practices such as evolutionary requirements, refactoring and minimal documentation. These issues can be addressed by further modifying the Agile RGM. This chapter provides an overview of an alternate structure to the Agile RGM that can accommodate large scale systems development.

One of the more important issues faced during the development of large scale systems (development period of more than two years) is accommodating changing requirements. *Requirements gathered for large scale systems change during the lengthy development periods due to changes in software and business environments, new user needs and technological advancements*. Agile methods which focus on accommodating changing requirements even late in the development lifecycle can be adopted for the development of large scale systems. However, these practices are not considered suitable for large scale systems development. The organizations involved in the development of large scale systems can be justified in not using agile practices because adopting them can have a detrimental effect on the system being developed. Some of the reasons for not adopting agile practices for large scale system development are listed below:

1. The practice of *evolutionary requirements* usually works very well for small and large scale projects equally [1]. However, this practice is not suitable if the system under development requires *Independent Verification and Validation (IV&V)*. IV&V is used in

the development of large scale and MLC systems to attain high reliability. As such, the requirements have to be gathered before the design phase in order to perform the Safety-Impact Analysis, which ensures that the safety of the system will not be compromised. Evolutionary Requirements, on the other hand, suggests that requirements should evolve over many iterations rather than being gathered upfront. Hence, adopting evolutionary requirements is not feasible when IV&V is used.

2. The agile approach uses *Refactoring* to continuously improve the code. "Code Refactoring involves rewriting the code to improve its structure, while explicitly preserving its behavior. In large scale systems, refactoring is a risky and costly task due to size and complexity of the systems" [1].
3. Refactoring is an essential component of Test-Driven Development which is adopted by many agile practitioners for implementing the features or stories. Since refactoring is risky in large scale systems, TDD is not considered suitable.
4. *"Working software over comprehensive documentation"* is one of the focal values of agile practices as specified in the agile manifesto [6]. Large scale systems usually have long development cycles and hence the issue of personnel turnover is inevitable. So, comprehensive documentation is required for training and support. Also, third-party organizations that perform the maintenance operations on such systems require documented information. Hence, minimal documentation can be detrimental to the development of large scale systems.

Nonetheless, agile practices such as evolutionary requirements (when IV&V is not used), pair-programming and direct stakeholder involvement have been proven to be successful for large scale system development [1]. Also, the agile methods propose an iterative and incremental approach to software development which helps deliver working software frequently. Hence, a hybrid approach combining the advantages of agile practices and structured methods would prove to be an effective solution to accommodating change in large scale systems. We propose the Generalized Agile RGM (Figure 5.1) which shows a hybrid approach that can be used for large scale system development. The Generalized Agile RGM is discussed in the next section.

5.1 Generalized Agile RGM

We propose the **Generalized Agile RGM Model** (Figure 5.1), which conveys a hybrid approach reflecting the best practices from both the agile and conventional approach to software development. The Generalized Agile RGM preserves the structure of the agile RGM and provides the flexibility to choose from existing structured approaches to software development to work within an agile framework. The rigidity of the process depends on the size of the project.

Agile methods focus on deciding the time line and resources for the project before estimating the features. This principle is followed in the Agile RGM and the Generalized Agile RGM. The developers are aware of the time frame for each release. The timeline for the development process is divided into release cycles and each release is made up of iterations which last for 2 – 4 weeks. Due to the size of the project under consideration, the release cycle may last for a period longer than 90 days. At the end of each release, working software of value to the customers is produced.

5.1.1 Generalized Agile RGM – Initial Phases

Large scale systems require a formal development process due to their size and complexity. Traditional software development approaches like the waterfall model which are adopted for large scale system development, suggest gathering user needs and transforming them into requirements. Often, information is lost during this transformation. Domain information in particular is not captured. Stories on the other hand, preserve the domain information. Also, customers create acceptance criteria for each story which provide information about the constraints on the system. These constraints can be mapped to requirements. The stories are validated by the stakeholders to ensure that the user needs and intents have been captured correctly. Hence, identifying stories before creating formal requirements preserves information. Features help identify the scope of the system. The generalized model preserves the Feature and Story Development phases because of the above mentioned reasons.

The first three phases of *Education, Feature Development and Story Development* of the Agile RGM are preserved in this model irrespective of the size of the project as shown in Figure 5.1 (a).

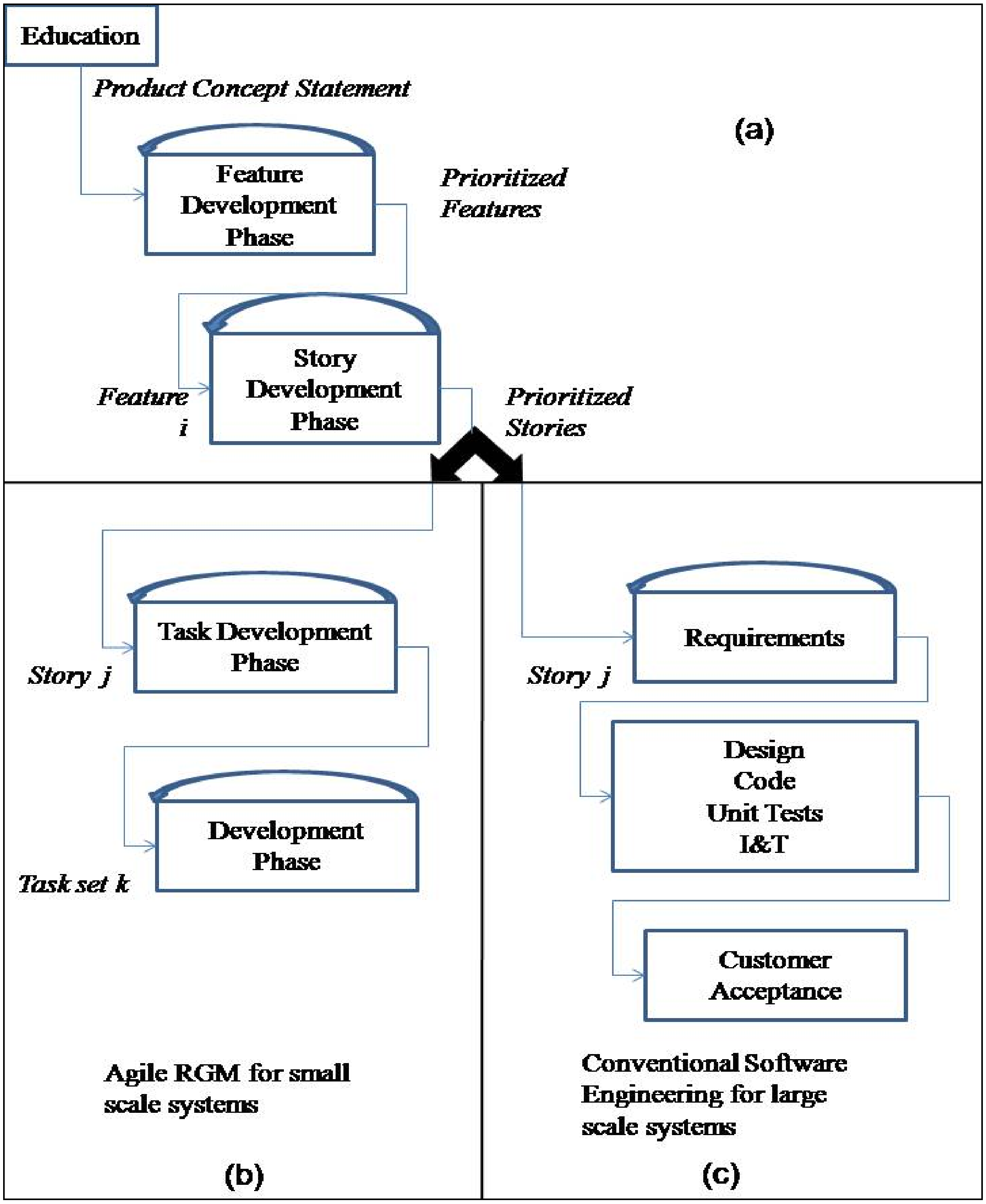


Figure 5.1.Alternate structure for large scale systems – Generalized Agile RGM

Initially, the stakeholders meet to establish rapport and the development team learns the business process of the customers. A high level mission statement called the Product Concept Statement is created. The high level features depicting the expected functionality of the system are identified from the product concept statement. These features are validated and the time required for their completion is estimated. These features are prioritized and stored in a prioritized story stack. Only a subset of the prioritized features is chosen for development during the current release cycle. Each feature to be implemented during a release cycle is then decomposed into stories which are descriptions of user- or customer-valued functionality [20]. One or more features can be decomposed into stories simultaneously. This reflects the JIT philosophy and helps accommodate change even late in the development lifecycle. These stories are validated and the time required for their completion is estimated. The stories are prioritized and stored in a prioritized story stack.

As shown in Figure 5.1, depending on the size and complexity of the system, the stories can be

a) decomposed into tasks and then implemented as in the Agile RGM (small scale systems)

or

b) converted into formal requirements and the requirements are implemented using a traditional approach like the waterfall model (large scale systems).

Case (a) is discussed in section 5.2.2 and (b) in section 5.2.3

5.1.2 Generalized Agile RGM – Small Scale Systems

When developing small scale systems, the Generalized Agile RGM can be represented by parts (a) and (b) of Figure 5.1. The output after the initial three phases is a set of prioritized stories for each feature. These prioritized stories for each feature are decomposed into tasks. Task lists are created for each story during the *Task Development Phase*. The tasks are then implemented during the *Development Phase* using TDD. The completed tasks are integrated with the existing code base. At the end of each iteration, the customers and the development team test the system available against the acceptance criteria created earlier. The task development and development phases of the Agile RGM are discussed in detail in Chapter 3.

5.1.3 Generalized Agile RGM – Large Scale Systems

As mentioned earlier, large scale systems development requires a structured approach. Parts (a) and (c) of Figure 5.1 shows that when developing large scale systems, a conventional approach can be adopted after the stories are developed. During the initial education phase, the stakeholders create a high level mission statement that is used to identify release level features during the Feature Development Phase. The features help determine the scope of the system. Each feature is decomposed into stories during the story development phase. The initial three phases of the Agile RGM are preserved.

Due to the size of the system being built, hundreds of stories may be created for the features and can result in story card hell. Hence, the stories should be converted into requirements to prevent chaos and ensure that information is not lost. ***Each story may be transformed into one or more requirements***.

As mentioned above, the prioritized stories are transformed into requirements. During an iteration, a subset of the prioritized stories is converted into requirements iteratively and incrementally. Here, the RGM or any formal RE approach can be used to guide the process of extracting requirements from user stories. Large scale systems require formal documentation. Hence, formal requirements specification documents can be produced.

As mentioned earlier, refactoring and TDD which are reflected in the Agile RGM are not suitable for building large scale systems. Hence, for building large scale systems, the *Task Development Phase* and *Development Phase* of the agile RGM have to be modified to accommodate a more structured process. The formal requirements produced during the requirements phase progress through ***formal design, code, unit tests, integration tests*** and finally ***customer acceptance tests***. Hence, for implementing each story, a ***"waterfall like"*** process is adopted. As we deal with requirements at the story level, it is easier to accommodate change even late in the development lifecycle. At each stage, formal artifacts like requirements specification and design documents can be created.

This process is repeated until the system all the features are identified and implemented. The Generalized Agile RGM is an incremental model. Hence, software is produced in increments. The development period spans multiple cycles and one or more features are implemented during a release cycle. The Generalized Agile RGM can support concurrency from the Story Development Phase onwards. If multiple teams are involved, parallel development efforts can be supported.

5.2 Summary

The Generalized Agile RGM combines agile practices like evolutionary requirements, no BRUF and JIT philosophy with a formal "waterfall" like approach. Other formal software development approaches like the spiral model and incremental model can be used instead of the waterfall approach to implement stories.

The model follows the JIT philosophy and the requirements are allowed to evolve over time. This helps accommodate changing requirements. The requirements are implemented in an incremental fashion. The Generalized Agile RGM can support concurrency from the Story Development Phase onwards. If multiple teams are involved, parallel efforts to create stories and then requirements can be undertaken. Concurrent development efforts can help reduce the time taken for the project completion.

This alternate structure as shown in parts (a) and (c) of Figure 5.1 is the Generalized Agile RGM which is a hybrid model that offers the advantages of the structured approaches to software development and the agile philosophy. This model can be customized according to individual system sizes and project needs. Hence, as the complexity of the system increases, more structure can be added to the model by adopting formal methods for the requirements, design, coding and testing phases.

The Generalized Agile RGM is also an independent model that can be integrated with any of the existing agile methods. The Generalized Agile RGM reflects the values and principles stated in the agile manifesto and hence, it can be used with other agile methods.

6. Conclusion

In this research, we present the Agile RGM, a soft-structured approach designed to guide and assist practitioners adopting agile RE. It was motivated by the lack of a structured approach to agile RE. The Agile RGM is designed for small scale systems (less than 1 year development period). The Agile RGM as an independent model that can be used with any of the existing agile methods. We also provide an overview of the Generalized Agile RGM, an approach that combines the best practices of the agile and conventional methods of software development. The Generalized Agile RGM is designed to accommodate large scale systems development. This chapter presents the main contributions of this research, a brief summary of this body of work and future opportunities for work related to the Agile RGM.

6.1 Contributions

The main contributions of this research were the creation of the Agile RGM and an overview of the Generalized Agile RGM. Both models are soft-structured approaches to agile software development. The time period for the complete development lifecycle of the product was considered when creating these approaches. Hence, these approaches span the complete development process from requirements gathering through design, coding, testing and customer acceptance. The Agile RGM and the Generalized Agile RGM are discussed briefly below.

Agile RGM

This research was motivated by the lack of a structured approach to agile RE. The specification of activities in agile RE is minimal and there is no mapping between the activities and the techniques that can be used to carry out these activities. We realized that these issues can be resolved by structuring the agile RE process by identifying a conventional RE approach and modifying it to suit an agile framework. The Requirements Generation Model (RGM) was chosen to be adapted to an agile environment. The RGM is a structured approach to capturing requirements. It includes all the major RE phases of requirements elicitation, analysis, specification, verification and management. The RGM is an iterative model which emphasizes on stakeholder involvement. These characteristics are reflective of the agile philosophy. Also, the RGM provides clear delineation of activities with adequate scope for modification. Hence, the RGM was chosen to structure the agile RE process. The Agile RGM was developed by combining the agile RE process and RGM.

The Agile RGM is an approach that structures the agile RE process. This was achieved by modifying the RGM to reflect agility. It is an iterative and incremental approach to agile software development for small scale systems (less than one year development period). It spans the complete development process and consists of five phases – education, feature development, story development, task development and development). The Agile RGM is an independent model that can be used with any of the existing agile methods. The approach supports concurrent development efforts.

Overview of the Generalized Agile RGM

The Agile RGM was designed for small scale systems (less than one year development period). It adopts TDD, refactoring and minimal documentation which are not considered suitable for large scale systems. However, the Agile RGM can be modified to allow practitioners to choose from existing structured approaches like the waterfall model depending on the size and complexity of the systems. The Generalized Agile RGM is a hybrid approach that can be used for the development of large scale systems. After creating stories for each feature, the practitioners can adopt a formal approach where each story is converted into formal requirements. These requirements progress through design, coding, testing and customer acceptance. As mentioned earlier, the practitioners can choose from any of the existing structured approaches to work within the Generalized Agile RGM framework.

6.2 Future Work

The Agile RGM is designed to add structure to the agile RE process. As the time period for development was also considered, the Agile RGM describes the complete development process. It describes activities to be carried out during each phase of the development process and suggests techniques that can be used. The following paragraphs briefly describe some of the possible future research efforts related to the Agile RGM.

Validate the Agile RGM by using it in an organization

The Agile RGM has not been used by any organization or team involved in software development. Using the Agile RGM independently or with other agile methods in an organization would help validate the approach. The feedback can be used to identify the strengths and weaknesses of the model. Also, changes can be made to model to ensure that it can work well in an organization.

Extend the Agile RGM to large scale and MLC systems development

The Agile RGM can be extended to the General Agile RGM to accommodate development of large scale systems. Currently, only an overview of the General Agile RGM (a hybrid approach to large scale systems development) is available and only the "big picture" is provided. The complete process shown by the General RGM can be described in detail. The activities in General RGM can be identified and specified clearly to accommodate large scale systems. The General RGM can be modified to accommodate the development of MLC systems. The suitability of the existing set of agile practices for MLC systems can be determined.

Developing a tool that supports the Agile RGM approach

A requirements management tool that supports the Agile RGM approach can be developed. Such a tool would help keep track of the user needs, features, stories and tasks. If distributed teams are involved, such a tool would enable sharing information among people in different locations.

6.3 Summary

This research involved the creation of the Agile RGM as a structured approach to the agile RE process. The Agile RGM is an independent model that can be used with any of the existing agile methods. This claim was substantiated by integrating the Agile RGM with Scrum, XP and Feature Driven Development. An overview of the Generalized Agile RGM that can be used for the development of large scale systems was also provided. This section provides a brief summary of the Agile RGM, the substantiation process employed and the Generalized Agile RGM.

Agile RGM

The creation of the Agile RGM was motivated by the lack of a formal process to the agile approach to requirements engineering. The Agile RGM is a soft- structured approach that serves as a guideline to the Agile RE process. It suggests practices and techniques that teams can adopt. The Agile RGM was created by adapting the Requirements Generation Model (RGM) [ ] to an agile development environment. The RGM is a conventional RE model and is a structured approach to capturing requirements. The RGM provides guidelines and protocols that practitioners can adopt to gather requirements.

The Agile RGM is designed for small scale systems using agile methods with 60 to 90 day release cycles and less than one year development period. As the length of the development lifecycle is taken into account, the Agile RGM describes not only the requirements engineering activities but the complete development process as well.

The Agile RGM consists of five phases which are briefly described below:

1. **Education phase** – This is a meeting which all the stakeholders (customers, users, developers, managers, domain experts, business analysts, testers, etc.) attend to establish rapport. The development team learns the business process of the customers. Also, a high level product concept statement which briefly outlines the problems to be addressed and the solution expected. The product concept statement is a high level mission statement which is modified when new requirements are identified.

2. **Feature Development Phase** – A feature is the smallest set of functionality that has business value to the customer [19]. The features are identified over a series of meetings and are defined at the highest level of abstraction. The features should be identified up front in order to recognize the scope of the system to be built. The identified features are validated and estimated. Customers prioritize the identified features based on their business needs. These prioritized features are stored in a prioritized feature stack. Priorities of the existing features are recomputed to accommodate the new features in the prioritized feature stack.

3. **Story Development Phase** – "Stories are descriptions of user- or customer-valued functionality" [21]. Each feature to be implemented in the current release is decomposed into stories. The rest of the features are decomposed and implemented during future release cycles. The stories are validated and estimated in story points. The developers prioritize the stories based on dependencies among them. They also consider customer and user preferences. The prioritized stories are stored in a prioritized story stack. If more than one feature is to be implemented in a release, multiple teams may work on identifying stories simultaneously.

4. **Task Development Phase –** Each story developed during the previous phase is decomposed into tasks. The task list for each story is essentially a to-do list created for the developers. Decomposing stories into tasks ensure that the developers do not overlook any detail while implementing stories. After the task lists are created, they are validated and estimated. Each task takes a few hours to complete.

5. **Development Phase –** Each task identified during the task development phase is implemented using Test Driven Development (TDD). After each task is completed, it is integrated with the existing code base. At the end of an iteration, the customers test the available system against the acceptance criteria created earlier.

The process continues over the future release cycles until the product is complete. The Agile RGM is an iterative and incremental model. Software is built in increments. The customers prioritize the features of the system under consideration. The prioritization is based on the business value of each feature. Concurrency is supported from the story development phase onwards.

Substantiation

The Agile RGM is an independent model that can be used with any of the existing agile methods. This claim can be substantiated by integrating the Agile RGM with agile methods like Scrum, XP, etc. Chapter 4 describes how the Agile RGM can be integrated with Scrum, XP and Feature Driven Development. The Agile RGM can work well with the other agile methods because it reflects the focal values and the principles stated in the Manifesto for Agile Software

Development. It also adopts practices specified by the agile approach to RE and software development like evolutionary requirements, JIT philosophy, TDD, refactoring, minimal documentation and direct customer involvement.

Generalized Agile RGM

One of the main issues faced during large scale systems development is accommodating changing requirements. Requirements change over the lengthy development cycles due to changes in software and business environments, new user needs and technological advancements. This issue can be solved by using agile methods for large scale systems development as the agile philosophy emphasizes accommodating changing requirements even late in the development lifecycle.

Agile methods are usually ignored by organizations involved in large scale systems development. Due to the size and complexity of the systems under consideration, practices such as refactoring, TDD and minimal documentation are deemed unsuitable. However, agile practices such as evolutionary requirements (when IV&V is not used), pair-programming and direct stakeholder involvement have been proved to be successful for large scale system development [ ]. Hence, a hybrid approach combining the advantages of agile practices and structured methods would prove to be an effective solution to accommodating change in large scale systems.

The Generalized Agile RGM presents such a hybrid approach that enables practitioners to use conventional software development practices within an agile framework depending on the size and complexity of the systems under consideration. Also, requirements are allowed to evolve over time. The Generalized Agile RGM preserves the initial three phases of the Agile RGM (education, feature development and story development). As large scale systems require a structured approach to their development, the stories can be converted to formal requirements using any conventional RE process. These requirements then proceed through design, coding, testing and customer acceptance.

References

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

[2] *IEEE Standards Collection: Software Engineering, I. S. 610.12-1990*.

[3] W.W. Royce, “Managing the development of large software systems: concepts and techniques,” *Proceedings of the 9th international conference on Software Engineering*, IEEE Computer Society Press, 1987, pp.

[4] B.W. Boehm, “A Spiral Model of Software Development and Enhancement,” *Computer*, vol. 21, no. 5, 1988, pp. 61-72

[5] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw - Hill International Edition, 2005.

[6] “Manifesto for Agile Software Development,” 2001; www.agilemanifesto.org.

[7] The CHAOS Report, Standish Group, 1995.

[8] F.P. Brooks, Jr., “No Silver Bullet Essence and Accidents of Software Engineering,” *Computer*, vol. 20, no. 4, 1987, pp. 10-19

[9] S.W. Ambler, “Requirements Envisioning: An Agile Best Practice,” http://www.agilemodeling.com/essays/initialRequirementsModeling.htm.

[10] S.W. Ambler, “Agile Requirements Modeling,” http://www.agilemodeling.com/essays/agileRequirements.htm.

[11] M.D. Alan, “The Art of Requirements Triage,” *Computer*, vol. 36, no. 3, 2003, pp. 42-49

[12] M.K. Groener, “Capturing Requirements Meeting Customer Intent: A Methodological Approach,” 2002 Ph.D. Dissertation, Virginia Tech ETD Collection, http://scholar.lib.vt.edu/theses/available/etd-05232002-234024/

[13] B. Boehm, et al., “Using the WinWin Spiral Model: A Case Study,” *Computer*, vol. 31, no. 7, 1998, pp. 33-44

[14] R. Jeffries, “What is Extreme Programming,” 2001; http://xprogramming.com/xpmag/whatisxp.htm.

[15] P. Abrahamsson, et al., *Agile Software Development Methods: Review and Analysis*, VTT Publications, 2002.

[16] S.R. Palmer and J.M. Felsing, *A Practical Guide to Feature Driven Development*, Prentice Hall PTR, 2002.

[17] K. Schwaber, “What is Scrum?,” 2007; http://www.scrumalliance.org/resources/227.

[18] J. Shore, “Beyond Story Cards: Agile Requirements Collaboration,” 2005; http://jamesshore.com/Multimedia/Beyond-Story-Cards.html.

[19] J. Patton, “Ambiguous Business Value Harms Software Products,” *IEEE Softw.*, vol. 25, no. 1, 2008, pp. 50-51; DOI http://dx.doi.org/10.1109/MS.2008.2.

[20] M. Cohn, *User Stories Applied: For Agile Software Development*, Addison Wesley Longman Publishing Co., Inc., 2004.

[21] K. Beck, *Test-Driven Development: By Example*, Addison-Wesley, 2003.

[22] R.C. Martin, *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall PTR, 2003, p. 710.

[23] W. Cunningham, “Fit: Framework for Integrated Test,” 2007; http://fit.c2.com/.

[24] “What is Fitnesse? ,” http://fitnesse.org/FitNesse.OneMinuteDescription.

[25] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*, Addison Wesley, 2003.

[26] J. Arthur, "Reconciling the Agile Philosophy with Plan-Driven Software Engineering: A “Lite” Approach to Requirements Engineering (Thoughts and Ruminations), " Presentation at Siemens Corporate Research Center, Princeton, NJ, August 2007

[27] H. Rex Hartson, CS 5714 Usability Engineering Class Notes, Virginia Tech, 2006 http://courses.cs.vt.edu/~cs5714/fall2006/Class%20notes/class%20notes.pdf