A Soft-Structured Agile Framework for Larger Scale Systems Development

Shvetha Soundararajan and James D. Arthur
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia, USA
{shvetha, arthur}@vt.edu

***Abstract*—One of the more important issues in the development of larger scale complex systems (product development period of two or more years) is accommodating changes to requirements. Requirements gathered for larger scale systems evolve during lengthy development periods due to changes in software and business environments, new user needs and technological advancements. Agile methods, which focus on accommodating change even late in the development lifecycle, can be adopted for the development of larger scale systems. However, as currently applied, these practices are not always suitable for the development of such systems. We propose a soft-structured framework combining the principles of agile and conventional software development that addresses the issue of rapidly changing requirements for larger scale systems. The framework consists of two parts: (1) a soft-structured requirements gathering approach that reflects the agile philosophy i.e., the Agile Requirements Generation Model and (2) a tailored development process that can be applied to either small or larger scale systems.**

Keywords- Managing Change; Small to Medium Scale Systems; Agile Software Development; Mitigating Development Complexities; Soft-Structured Processes

I. INTRODUCTION

Currently, the number of organizations adopting agile practices is increasing - some of the reasons being (1) the ability to accommodate changes to requirements, (2) enhanced customer relationships, (3) greater return on investment and (4) shorter development periods. In spite of these advantages, adoption of agile methods for the development of larger scale systems is minimal. Moreover, if such systems are ill equipped to accommodate change, additional complexity can be introduced during the maintenance activities. Hence, larger scale systems require a more structured development process that can accommodate that change.

Agile practices such as code refactoring, minimal documentation, etc., are not always suitable for the development of larger scale systems. For example, the agile philosophy insists on minimal documentation when building a software system. The focus is on producing working software rather than comprehensive documentation. However, comprehensive documentation is often necessary when a larger scale system is under consideration in order to provide third party maintenance organizations with documented information. Also, larger scale systems usually have lengthy development cycles. Subsequently, personnel turnover also demands some form of software documentation. Hence, comprehensive documentation is required for support and training.

On the other hand, agile practices such as evolutionary requirements, pair-programming and direct stakeholder involvement have been proven to be successful for larger scale systems development [1]. Agile methods propose an iterative and incremental approach to software development, which helps deliver working software at regular intervals. The rationale behind these agile methods and practices is to accommodate change. Also, agile methods are lightweight and the focus is on the people rather than the process. However, in opposition to this philosophy, the development of larger scale systems requires a structured approach. Thus, a hybrid approach combining the advantages of agile practices and structured methods can be an effective solution to accommodating change in larger scale systems. The challenge in developing such a hybrid approach is that the impact on agility should be minimal.

In this paper, we propose a soft-structured approach for the development of larger scale systems. This approach accommodates change because it reflects the agile philosophy. By soft-structured, we mean structuring the software development process somewhat, but at the same time providing practitioners with the flexibility to employ many of the existing agile practices. Our hybrid approach reflects the philosophies of both agile and conventional approaches to Software Engineering. The framework consists of two parts:

1. The Agile Requirements Generation Model, which is a soft-structured approach to gathering requirements, and

2. A development process that provides alternative approaches based on the type of system (small or larger scale) under consideration.

These parts are briefly explained in the next two paragraphs.

Our objective has been to evolve an approach to developing larger scale systems that can accommodate change. The first step in achieving this goal is to ensure that the requirements gathering process is flexible and can accommodate change. We have chosen to embrace the principles of Agile Requirements Engineering (Agile RE) [2], and apply the embodying philosophy to the conventional RE process. The objective behind Agile RE is to accommodate changes to requirements. However, in its current state, Agile RE is relatively unstructured. That is, the specification of activities is minimal, and the mapping between the activities and the techniques that can be used to carry out these activities is limited. Moreover, our understanding of Agile RE is, more often than not, tacit. We contend that these issues can be resolved by structuring the Agile RE process by identifying a conventional RE approach and modifying it somewhat to reflect an agile environment. In this work, we tailor the Requirements Generation Model (RGM) [3], a conventional requirements gathering approach to reflect the agile philosophy. The result is the Agile Requirements Generation Model (Agile RGM) [4] that (1) is an iterative and incremental approach to gathering requirements and (2) reflects the values of agility.

The second part of the framework is the development process that incorporates the requirements gathered using the Agile RGM. Depending on the type of system under consideration (small or larger scale), we can adopt a development process leaning towards either an agile or a more conventional approach. We propose two alternative development approaches because there is no "one size fits all" solution to software development. That is, conventional "waterfall like" Software Engineering approaches may be onerous to teams involved in building small-scale systems. On the other hand, larger scale systems require a structured approach. In order to embrace both types of systems, we present two alternative paths in our development process.

The main objective of our soft-structured approach is to accommodate change. Hence, both the requirements gathering approach and the development process reflect the agile philosophy.

Section 2 of this paper provides some background information about Agile RE and the RGM. The two parts of the framework discussed in this paper are presented in Sections 3 and 4. More specifically, Section 3 discusses the Agile RGM and Section 4 the development process. We present the suitability of the soft-structured for larger scale systems development in Section 5. In Section 6, we substantiate the effectiveness of the framework. Section 7 summarizes our research.

II. Background

The main objective of the research presented in this paper has been to propose a Software Engineering approach that preserves agility and can help address the change-induced complexities associated with larger scale systems development. In this paper, we describe a soft-structured framework that spans the Agile RGM and our tailored alternative approaches to the development process. Because both build on the Agile RE philosophies and the RGM, we provide a brief discussion of each in the following two sub-sections.

A. Agile RE

The agile principles applied to Software Engineering include iterative and incremental development, frequent releases of software, direct customer involvement and minimal documentation. All of the above are designed to accommodate change. The current agile approach to RE applies these principles to the RE process.

Conventional RE processes focus on gathering all the requirements upfront and preparing the requirements specification document before proceeding to the downstream development phases. These upfront requirements gathering and specification efforts leave little room to accommodate changes to requirements identified during the design or coding phases of software development. On the other hand, Agile RE welcomes changing requirements. This is achieved by using the agile practice of *Evolutionary Requirements*, which suggests that requirements should evolve over the course of many iterations rather than being gathered and specified upfront. More specifically, the high-level features for the system, which represent expected functionality, are identified first. The details for each feature are then gathered "just-in-time" (JIT) from the stakeholders, and right before the development of that feature. That is, customer and user needs are elaborated and refined as and when required. This is the just-in-time philosophy and is one of the focal values embraced by agilists. Stakeholders are actively involved in the Agile RE process. Changes to requirements identified are logged and are implemented in the subsequent iterations.

Agile RE also focuses on minimal documentation. No formal requirements specification is produced. The features and the requirements are recorded on storyboards and index cards. The artifacts produced depend on the project. Example Agile RE artifacts include paper prototypes, use case diagrams and data flow diagrams.

Verification and Validation (V&V) of requirements is an important activity for any RE approach. However, there is no explicit specification of V&V activities in Agile RE. Because the customer is usually available onsite, the features/requirements can be validated in a just-in-time fashion. Moreover, if and when verification criteria are stated, they are usually in the form of user stories. Hence, verification is more of a validation process.

The current state of Agile RE is that it provides principles and practices, but no standard process for applying them. In short, there is only minimal process specified in the Agile RE approach. In the research presented in this paper, we have structured the Agile RE approach by using the Requirements Generation Model (RGM) described next.

B. RGM

We have adopted the RGM [3] to structure the Agile RE process because it is flexible and reflects agility. The RGM is a structured approach to capturing requirements. It covers all of the activities of the requirements engineering process, namely requirements elicitation, analysis, specification, verification and management.

The RGM is an iterative process as can be inferred from Figure 1. Each iteration through the RGM produces a set of requirements. Requirements are produced in increments. The RGM focuses on Big Requirements Up Front (BRUF). That is, all of the requirements are identified upfront before proceeding to the next phase in the Software Engineering lifecycle. It encourages collaboration among the stakeholders. These attributes of the RGM are highly reflective of the agile philosophy. In addition, this model provides well-defined activities for the different phases of the RE process. Therefore, we chose this approach to structure the Agile RE process.

As shown in Figure 1, the RGM process starts with the indoctrination phase, which serves as the education component for both developers and customers. The preparation phase is a meeting among the project stakeholders to determine the scope of the following elicitation phase. The elicited requirements are then reviewed in the evaluation phase. The need for additional iterations of the RGM is also determined at this time.

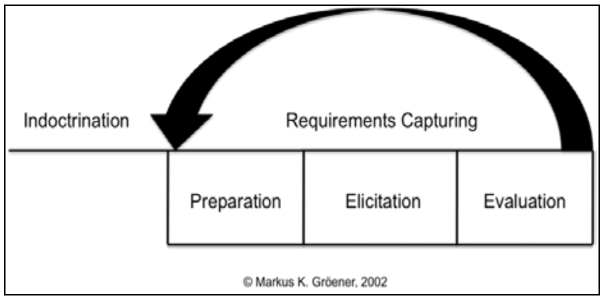

Figure 1. Requirements Generation Model (RGM)

The Agile RGM is designed by adapting the RGM to reflect more directly the Agile RE philosophy. Agile RE focuses on applying agile principles and practices to the RE process. The RGM is reflective of the agile philosophy and we modified it to suit an agile software development environment.

As previously mentioned, our approach to addressing change-induced complexities stemming from the development of larger scale systems involves a synergistic coupling of two components: the Agile Requirements Generation Model and an appropriately tailored software development process. Both of these are discussed more fully in the following two sections.

III. Agile Requirements Generation Model (Agile RGM)

The main objective of this work has been the development of a soft-structured approach to facilitate larger scale systems development that can fit within an agile framework. The approach embraces agility in order to accommodate changes to requirements. The Agile RGM [4] is the initial component of the approach outlined in this paper. In this section, we present the Agile RGM as a soft-structured approach to Requirements Engineering that embraces agility.

The main objective of the Agile RGM is to accommodate change. It specifies a set of well-defined activities that provide a more structured approach to gathering requirements. Its soft-structured characteristic provides the practitioners with the flexibility to adopt practices and techniques suitable to their teams and organizations. The Requirements Generation Model described in Section 2.2 provides that necessary structure for the Agile RE process, and thereby helps guide the practitioners. Within this soft-structured approach to Requirements Engineering, we have incorporated the Agile RE principles and practices such as direct stakeholder involvement, evolutionary requirements, refactoring, no BRUF, just-in-time gathering of details and minimal documentation.

Figure 2 shows the Agile RGM. This forms the initial component of the soft-structured framework for engineering larger scale systems.

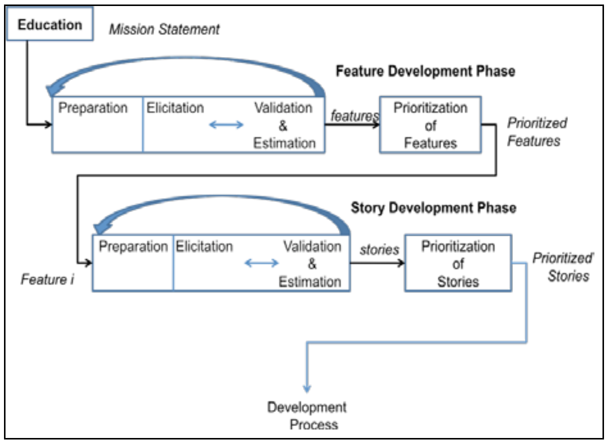

Figure 2. Agile RGM

As shown in Figure 2, there are three main phases in the Agile RGM: Education, Feature Development and Story Development. The process begins with the Education phase where the development team acquires a better understanding of the business process of the customers. Additionally, a high-level mission statement describing the problem to be addressed and an outline of the solution to be implemented is created.

The mission statement helps identify the expected system functionality during the Feature Development phase. The customers and the development team determine the system features that are of value to the customer. These features are implemented in an incremental fashion through multiple release cycles. The features to be implemented during a specific release cycle are then decomposed into stories in the Story Development phase. The stories describe the features in greater detail and are created just-in-time. The output of the Story Development phase is a set of prioritized stories, which serve as the input to the Development process described in Section 4. Although not explicitly illustrated in Figure 2, the decomposition of features into stories can occur concurrently. The phases of the Agile RGM are described briefly in the following paragraphs.

Education phase

The Education phase is the first step in the process outlined by the Agile RGM as shown in Figure 2. This phase is essentially a meeting among the various project stakeholders (stakeholders include business analysts, customers, users, developers, project managers and testers). The main objective is for the development team to gain a better understanding of the business process of the customers. This is essential in order to obtain the necessary domain knowledge. The stakeholders also create a high-level mission statement, which identifies the problems faced by the customers, a solution outline, and the users of the system. The solution outlined in the mission statement helps determine the expected functionality during the Feature Development phase described next.

Feature Development phase

The mission statement created during the Education phase serves as the input to the Feature Development phase. The stakeholders iteratively identify the expected system functionality (features) from the mission statement. A feature can be defined as the smallest set of functionality that provides business value to the customer [5]. "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)" [6]. For example, consider the development of a website for an e-commerce company. A feature of business value to the company would be "Online Payment". This feature implies that users of this website can complete their financial transactions online. This is of value to the e-commerce company and the user visiting the website. The identified features are stated at the highest level of abstraction.

As shown in figure 2, the activities of the Feature Development phase are Preparation, Elicitation, Validation and Estimation and Prioritization. These well-defined activities span the phases of a conventional RE process. These activities and the iterative nature of the Feature Development phase mirror the RGM described in Section 2.2.

The process begins with the Preparation activity. The objective of this activity is to set up or pre-determine a time for eliciting the features. That is, preparation is a meeting to plan for gathering features.

The Preparation activity is followed by Elicitation. Elicitation of features can take the form of brainstorming sessions, open-ended interviews, and focus groups, etc. Features can be recorded on index cards, white boards, electronic cards, etc.

The customers then validate each identified feature. Because the customers and the users are directly involved throughout the process, just-in-time validation of features is possible. We have included an explicit Validation activity in each phase of the Agile RGM (Figure 2) to ensure that the stakeholder needs and intents are correctly captured. Elicitation and Validation activities can take place synchronously as indicated by the horizontal arrow between these two activities shown in Figure 2. Finally, after each feature is validated, the developers estimate the time required for the completion of each feature.

As also shown in Figure 2, a Prioritization activity follows the iterative component in the Feature Development phase. The customer prioritizes 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 referred to as a prioritized feature stack.

Only one feature (or a subset of the prioritized 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 future release cycles.

Features are elicited over multiple iterations. The stakeholders strive to identify as many features as possible before proceeding to the Story Development phase. However, if a feature is identified late during the development life cycle, the time required for its completion is estimated, its priority is determined and it is subsequently added to the prioritized feature stack. On identifying new features, existing priorities should be reassessed.

Story Development phase

As mentioned earlier, features are prioritized based on their business value and are implemented in an incremental fashion. Features are stated at the highest level of abstraction, but the developers require additional details before proceeding to the development process. Hence, each feature chosen for implementation during a current release is decomposed into stories. Stories represent *refined user-*

or customer- expected functionality. Consider the example discussed previously. We mentioned "Online Payment" as a feature of business value to an e-commerce company. A story for the above-mentioned feature can be "As a user, I can pay by credit card". It represents a user's expectation from a feature supported by the system being developed.

The stories for each feature are identified over a number of iterations and are then validated, estimated and prioritized. As can be inferred from Figure 2, the activities of this phase mirror the Feature Development phase. The prioritized stories are stored in a prioritized story stack. These stories are then implemented during the Development Process of our soft-structured approach presented in this paper.

If multiple teams are involved in the development of the system, each team can work independently towards decomposing one or more features into stories.

The prioritized stories created using the Agile RGM are implemented during the development process. As mentioned earlier, the development approach can be chosen based on the type of system being built. These two approaches are discussed next.

IV. DEVELOPMENT PROCESS

The Development Process described in this Section forms the second component of the soft-structured framework for developing larger scale systems. We provide two alternative approaches to the development process depending on the type of the system under consideration. As shown in Figure 3a, if the system to be built is small-scale (development period of one year or less), we propose the decomposition of previously identified stories into tasks and then the implementation of these the tasks using Test Driven Development (TDD) [7]. On the other hand, if the system under development is a larger scale system (development period of two years or more), then the prioritized stories created by using the Agile RGM, are implemented using a more conventional, "waterfall like" approach (see Figure 3b).

We propose an alternative approach for larger scale systems because Agile practices such as TDD and refactoring are not considered suitable for larger scale systems development. This is due in part to the size and complexity of such systems. The next two subsections describe the two approaches that form the Development component of our soft-structured framework.

A. Development process – Small-scale systems

For small-scale systems, the prioritized stories from the Story Development phase are implemented using the approach shown in Figure 3a. Using this approach, we advocate using a Task Identification phase followed by TDD to implement the expected functionality. These activities are discussed in the following paragraphs.

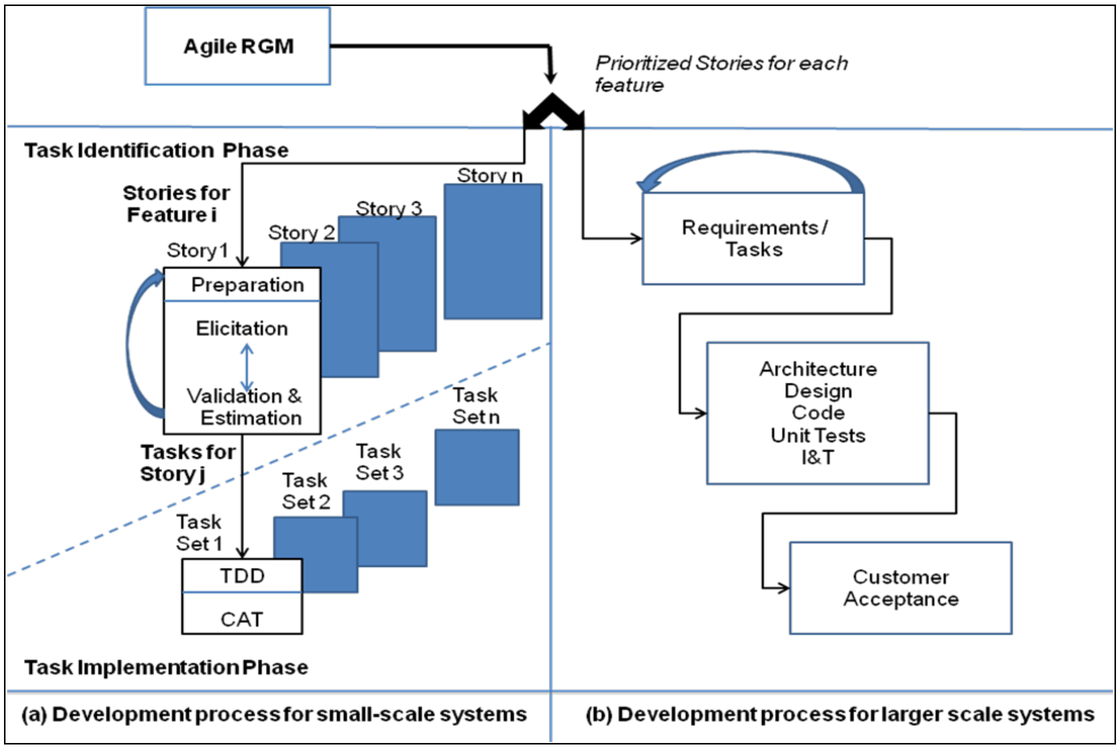
Agile RGM
Prioritized Stories for each feature
Task Identification Phase
Stories for Feature i
Story 1
Story 2
Story 3
Story n
Preparation
Elicitation
Validation & Estimation
Tasks for Story j
Task Set 1
Task Set 2
Task Set 3
Task Set n
TDD
CAT
Task Implementation Phase
Requirements / Tasks
Architecture Design Code Unit Tests I&T
Customer Acceptance
(a) Development process for small-scale systems
(b) Development process for larger scale systems

Figure 3a. Development Process for small-scale systems. Figure 3b. Development Process for larger scale systems

Task Identification phase

Initially, the prioritized stories for each feature are decomposed into tasks. The stories chosen for development during the current iteration serve as the input to the Task Identification Phase. Each story is decomposed into tasks by the development team. The Task Identification 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 [8]:

- Each story may be developed by more than one developer due to time constraints or developer skill sets. Therefore, there is a need to further decompose stories into tasks.
- Decomposing stories into tasks ensures that the developers do not overlook necessary details.

The task lists for each story are created during one or multiple iterations of the Task Identification 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. For the example story "As a user, I can pay by credit card" discussed in the previous Section, the task list could be as below:

1. Elicit credit card details
2. Verify order details
3. Authorize credit card information
4. Ensure that only Visa and Master cards are accepted
5. Display order confirmation

These tasks help ensure that no detail is overlooked.

The activities of the Task Identification Phase are similar to those described in the Feature and Story Development phases.

Task Implementation phase

As shown in Figure 3a, the identified tasks are then passed on to the Task Implementation Phase where the tasks are implemented using TDD and tested. Using TDD, developers create tests first before writing code. The developed code is then refactored to improve its structure. The rule is to write operational code only if a test fails. Delivering a product of value to the customer is a fundamental agile principle and hence, Customer Acceptance Testing (CAT) is of great importance. 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 that augment those written by the customers. The Agile RGM and the development process outlined in this paper reflect the agile philosophy. Hence, we suggest using TDD and CAT as activities for the Task Implementation phase.

Each task created earlier is implemented 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.

Concurrency in the Agile RGM

Concurrency is supported in the soft-structured framework discussed in this paper. More specifically, concurrent Story Development, Task Identification and Task Implementation efforts are feasible. Figure 3(a) shows concurrency in the Task Identification and Implementation phases. More than one developer can work on the tasks created for each story. Each developer chooses a set of tasks for the story to be implemented based on their availability and skill set. After completion, each task is integrated into an existing code base. Each iteration yields working software.

The advantage of concurrency is that it supports rapid development of software. The downside to increased concurrency is that the extent to which the just-in-time philosophy can be adopted is reduced. Let us assume that stories 1, 3 and 5 are being refined concurrently. Changes to story 1 can affect stories 3 and 5. As stories 3 and 5 are being refined concurrently, it would involve more time and effort to ensure that the changes made are consistent. This reduces the degree to which change can be accommodated. Hence, supporting concurrency limits the advantages of adopting the just-in-time philosophy.

The Agile RGM described in Section 2, together with the development process outlined above, provide a soft-structured approach to Software Engineering for small-scale systems. This approach helps accommodate changes to requirements even late in the development lifecycle.

B. *Development Process – Larger scale systems*

As discussed previously, larger scale systems require a more structured approach. The Agile RGM described earlier coupled with the approach shown in Figure 3b, provide a more conventional approach that can be adopted after the stories are developed. Due to the size of larger scale systems, hundreds of stories may be created for the features and can result in "story card hell" [5]. Hence, the stories should be converted into requirements to prevent chaos and to ensure that information is preserved. Subsets of stories can be transformed into one or more requirements.

In our approach to developing larger scale systems, subsets of prioritized stories are converted into requirements iteratively and incrementally. Consider, for example, the story: "As a user, I can pay by credit card." As shown in Section 4.1, this story is decomposed into five tasks. No further decomposition is required. From a requirements perspective, however, a much more detailed, definitive, and substantially larger set of specifications is required. One requirement generated by this story might be "The system shall use the Advanced Encryption Standard (AES) to encode all credit card information to be transmitted over the internet." Note the level of specificity and testability embodied in this (and reputedly all) requirement(s). It is this required level of detail and the necessarily restricted interpretation latitude that differentiates story decomposition within a strictly agile environment from that found in a more conventional development process. The RGM or any conventional Requirements Engineering approach can be used to guide the process of identifying requirements from user stories and thereby producing a formal specification of requirements.

The requirements produced during the requirements phase progress through *architecture, design, code, unit tests, integration tests* and finally *customer acceptance tests* (Figure 3b). Hence, for implementing each story, a *"waterfall like"* process is adopted. Although we adopt a conventional approach here, it fits within an agile environment as guided by the features identified early in the process. Even though we now have requirements, it is still easier to accommodate change because these requirements are derived just-in-time from stories.

As a final observation, we would like to note that refactoring (or code restructuring) is an important practice within agile development environments. Because the agile approach promotes change tolerance, enhancing a system's design through refactoring, *as it is being built*, is encouraged. Within the more conventional development approaches (like those for larger scale systems), such change is discouraged. More specifically, the architectural and detailed designs are determined *prior* to coding, and are not intended to change. Clearly, this limits development flexibility, but also minimizes the potentially detrimental impact and ripple effect of change in larger scale systems. Nonetheless, because our soft-structured agile approach focuses on *independent* feature development, that change restriction is primarily limited to the feature(s) currently under development.

The Agile RGM with the development process proposed in this subsection provides a complete Software Engineering lifecycle for larger scale systems. As we have integrated a conventional approach within an agile framework, accommodating changes to requirements is feasible even late in the development process. Hence, we now have an approach to developing larger scale systems that can accommodate change.

V. Suitability of the Soft-Structured Framework for Larger Scale Systems Development

Figure 4 shows the spectrum of Software Engineering approaches and their suitability for larger scale systems development. Our framework defines a middle ground between agile methods like eXtreme Programming (XP) [10] and conventional Software Engineering approaches like the waterfall model. Our framework provides structure to the agile approach to Software Engineering and at the same time avoids constraining the practitioners like in the conventional methods.

The main objective of the framework presented in this research is to provide an approach for both small and larger scale systems development and to accommodate change in both. Hence, the framework outlined in this paper is a hybrid approach combining the advantages of both Agile and structured methods for Software Engineering.

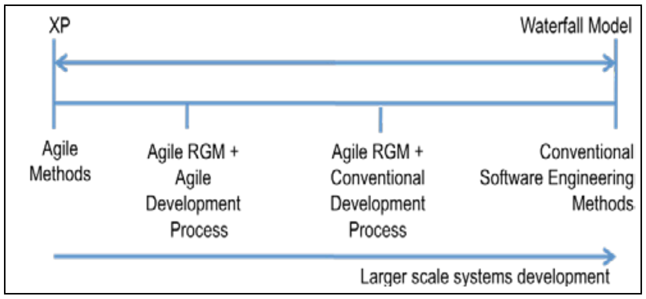
XP
Waterfall Model
Agile Methods
Agile RGM + Agile Development Process
Agile RGM + Conventional Development Process
Conventional Software Engineering Methods
Larger scale systems development

Figure 4. Spectrum of Software Engineering approaches for larger scale systems development

We also understand that not all teams and organizations have the need for a highly structured approach to software development. Conventional approaches may be onerous for small-scale systems. Therefore, we provide two alternative approaches (see Figures 3a and 3b) that can be adopted for implementing the "requirements" for the system under consideration. The initial feature/ story identification component using the Agile RGM is common to both the development approaches presented here. This phase introduces aspects of agility *independent* of project size. Then, depending on the size of the system to be built, we adopt a more Agile or conventional approach to implementing the desired functionality.

Our approach is suitable for larger scale systems as we provide a semi-structured approach that can accommodate change. On a scale of one to ten, with one being the most agile process and ten being the conventional waterfall-like approach to software development, we estimate that our approach for small-scale systems merits a three and larger scale systems a six (Figure 4).

VI. Substantiation of the "Goodness" of the Soft-Structured Framework

We substantiate the "goodness" of our approach by showing that it reflects (1) the values of the agile community and (2) the values of the industry. We first establish that our approach fits within an agile environment by showing that it reflects the principles stated in the Agile Manifesto and those of Agile RE. We also show that it can be integrated with the existing agile methods. Secondly, we briefly state our perception of feedback obtained when we presented our soft-structured approach to a development organization.

A. Agile Community

The agile community values the principles stated in the Agile Manifesto. It also endorses agile methods such as XP [10], Scrum [10], Feature-Driven Development [11], etc. In this Section, we discuss how our approach reflects the principles and practices embraced by the agile community.

Agile Manifesto

The framework described in this paper can be used with the existing agile methods. This is feasible as it reflects the values and principles stated in the Agile Manifesto. The Agile Manifesto states that the customers and the development team should be prepared to accommodate change even late in the development lifecycle. This is conveyed by the fourth focal value "*Responding to change* over following a plan" [9]. Our framework adopts the just-in-time philosophy that helps accommodate change, to gather and implement customer and user needs. There are no upfront planning activities specified.

Each feature identified using the Agile RGM is refined and decomposed into stories right before its implementation. Subsequently, each story is decomposed into tasks in a JIT fashion. Hence, changes to features and stories can be more easily accommodated. Additionally, developers estimate the time required for implementing features, stories and tasks just-in-time. Since no plans are created upfront, it is easier to adjust the amount of work to be completed in a given time frame. Hence, our approach focuses on responding to change rather than following a plan. Similarly, it also reflects the other focal values stated in the manifesto.

Agile RE

Agile RE is adopted in order to accommodate changes identified during the later phases of software development. Our approaches to both small and larger scale systems development reflects the principles and practices of Agile RE and, in turn, can accommodate change.

Evolutionary Requirements is an agile practice that states that requirements should evolve over time. In our approach, the stakeholders identify features initially to determine the scope of the system. Only a subset of the identified features is decomposed into stories, which in turn are decomposed into tasks. Hence, the requirements are not identified upfront. They *evolve* over time. The framework also adopts practices such as just-in-time gathering of details, direct customer involvement and minimal documentation, which are also reflective of Agile RE. We have also introduced explicit V&V efforts throughout the process to ensure that the requirements are correctly captured.

Agile Methods

Scrum [10] is an agile approach to managing the software development process. It employs an iterative and incremental process skeleton that includes a set of pre-determined practices and roles. However, it does not provide implementation techniques. It is used with other widely adopted agile methods such as XP [10], Feature-Driven Development [11], Crystal [12], etc., in order to provide a complete software development process. All of these agile methods suggest an iterative and incremental approach to Software Engineering. More specifically, these methods outline a "scrum-like" process. Therefore, they fit right within the Scrum skeleton. Our approach described in this paper advocates a similar "scrum-like" process. In the following paragraphs, we show that our approach can be integrated with Scrum. This in turn shows that our approach reflects the principles of the existing agile methods.

The Education and Feature Development Phases can be made a part of an initial Scrum meeting often called Scrum 0 because both of 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. For small-scale systems, during a sprint, activities described in the Story Development, Task Identification and Implementation phases of our approach for small-scale systems can be carried out. At the end of each sprint, a potentially shippable product increment is produced.

Similarly, if the system under consideration is a larger scale system, each sprint cycle can be considered as a "mini waterfall like" process. Each sprint cycle would involve creating stories and mapping subsets of stories into requirements. Each requirement would then proceed through formal design, code, test and customer acceptance phases as outlined by our approach for developing larger scale systems.

Using our approach within a Scrum process can serve as a guide 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 our approach. Hence, no additional effort is required to create them.

B. Industry

This work was also presented at Capital One, Richmond, Virginia and was well received. We found that our approach reflects many of the principles/ values embraced by that organization. We have mentioned the idea of using them as a beta-site for testing our approach on some of their projects. This would serve as a formal validation approach for our work.

VII. Conclusion

Our work has been motivated by the need to address the issue of accommodating change when developing larger scale systems. To achieve this objective, we propose a soft-structured approach to engineering larger scale systems while still preserving the desirable benefits of agility. Our approach accommodates changes to requirements even late in the development lifecycle. We also recognize the need for adopting different implementation approaches. Hence, we provide two alternative paths for implementing the expected system functionality. Our approach is purposefully designed to provide practitioners with the freedom to choose Software Engineering practices based on their needs. Our next step is to validate the applicability of the Agile RGM through an empirical study using a real life project. More specifically, we plan to incorporate our process in an organization and study its effectiveness.

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* (SEW 2007), IEEE Computer Society, 2007, pp. 3-12, doi: 10.1109/SEW.2007.61.

[2] S. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, John Wiley & Sons, New York, 2002.

[3] J. D. Arthur and M. K. Groener, "An operational model for structuring the requirements generation process", *Requirements Engineering*, vol. 10, no. 1, Jan. 2005, pp. 45-62, doi: 10.1007/s00766-004-0196-2.

[4] S. Soundararajan, "Agile requirements generation model: A soft-structured approach to agile requirements engineering", master's thesis, Dept. of Computer Science, Virginia Tech, 2008; http://scholar.lib.vt.edu/theses/available/etd-08132008-193105/.

[5] J. Shore, "Beyond story cards: Agile requirements collaboration", 21 Mar. 2005; http://jamesshore.com/Multimedia/Beyond-Story-Cards.html.

[6] J. Patton, "Ambiguous business value harms software products", *IEEE Softw.*, vol. 25, no. 1, Jan. 2008, pp. 50-51, doi: 10.1109/MS.2008.2.

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

[8] M. Cohn, *User Stories Applied: For Agile Software Development*, Addison-Wesley, 2004.

[9] "Manifesto for Agile Software Development", 2001; www.agilemanifesto.org.

[10] P. Abrahamsson, O. Salo, J. Ronkainen and J. Warsta, *Agile Software Development Methods: Review and Analysis*, VTT Publications, Finland, 2002.

[11] S. R. Palmer and J. M. Felsing, *A Practical Guide to Feature Driven Development*, Prentice Hall PTR, Upper Saddle River, New Jersey, 2002.

[12] A. Cockburn, *Agile Software Development*, Addison-Wesley, 2002.