Towards Tool Support for Situational Engineering of Agile Methodologies

Zahra Shakeri Hossein Abad, Mahsa Hasani Sadi, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
E-mail: z_shakeri@ce.sharif.edu, mhsadi@ce.sharif.edu, ramsin@sharif.edu

***Abstract*—Various agile software development methodologies, practices, and techniques have been proposed in the last decade; some present novel ideas, while many are simply made up of tasks and techniques borrowed from prominent agile methodologies. Each of these methodologies prescribes a set of practices and techniques which are deemed appropriate for application in a specific context. However, there exists no single method which fits all project situations. This has resulted in the advent of Situational Method Engineering (SME) approaches, which are used for developing software methodologies that are tailored to fit the specific circumstances of the project situation at hand. Since tool support has become an essential prerequisite for widespread adoption of software engineering methods, provision of Computer-Aided Method Engineering (CAME) tools has become a priority. We provide a basis for the application of assembly-based situational method engineering to the development of bespoke agile methodologies. To this aim, a comprehensive set of relevant methodology features has first been identified, spanning the range of possible requirements that a method engineer may define for the agile methodology under development. Based on this set of requirements, a method base has been proposed that contains the method chunks necessary for satisfying these requirements. The proposed method base conforms to the Software Process Engineering Metamodel (SPEM 2.0), and can be immediately plugged into CAME tools which implement this metamodel, including the Eclipse Process Framework Composer (EPFC).**

Keywords-agile software development methodology; situational method engineering; tool support; methodology requirement

I. INTRODUCTION

Since their emergence on the software engineering scene, agile methodologies have won widespread acclaim among managers and developers. This popularity is mainly due to the set of considerations that agile methodologies take into account in their processes – as stated in the Agile Principles [1] – including frequent delivery of executable products, active user involvement in the development process, and acceptability of change even if requested late in the development process. Based on these principles, a myriad number of techniques, practices, activities and processes have been introduced, both in practice and in the literature. Although several prominent agile processes exist – such as DSDM [2], Scrum [3], XP [4], ASD [5], Crystal Clear [6] and FDD [7] – they do not completely meet the situation-specific needs of method engineering projects.

In order to address the issue of developing and/or tailoring agile processes according to the requirements of a specific project situation, it has become essential to apply Situational Method Engineering (SME) approaches [8]. However, the introduction of a solitary SME approach does not solve the problems confronted in practice. A solid practical framework is needed that is applicable to a wide range of projects through adequate tools. Tool support has thus become indispensable, especially through specialized CAME (Computer-Aided Method Engineering) tools [9] that support the development of agile methodologies. On the other hand, the sheer number and variety of agile practices and processes poses a serious problem for methodology engineers: they have to choose from a vast pool of process components at various levels of granularity and with widely differing features. Aiming at resolving these issues, we have explored the utilization of method engineering approaches for developing agile methodologies, and have developed the basis required for the provision of full CAME tool support in this context.

Among the different strategies of SME, *assembly-based* SME has received considerable attention, and has become the basis of method construction in CAME tools. This strategy, in which a method is composed of reusable components called method chunks, defines the following four steps for method construction [8]:

1. Specification of the situation of the project at hand.
2. Definition of methodology requirements which reflect the project situation.
3. Selection of a set of method chunks satisfying the requirements.
4. Assembly of the selected method chunks into a coherent methodology.

Following the above steps, four preliminary prerequisites should be fulfilled in order to provide adequate CAME tool support for assembly-based engineering of agile methodologies:

1. Specification of a set of methodology requirements that are of main concern in the development of agile software processes.
2. Extraction of a set of agile process fragments from existing agile methodologies. These fragments are in fact empirically proven patterns of practice at different

1530-1362/10 $26.00 © 2010 IEEE
DOI 10.1109/APSEC.2010.45

levels of granularity applied in the context of agile development projects.

3. Matching the extracted method chunks with the specified methodology requirements which they satisfy, thereby forming a cohesive set of method chunks.
4. Supplementing the method chunks with relevant guidelines on how they can be assembled into a coherent process.

The first three stages mentioned above provide a comprehensive repository of method chunks, referred to as the method base (or method repository) in CAME tools [9]. We propose an agile method base as the first necessary step in assembly-based engineering of agile methodologies. However, since the ultimate goal is to provide CAME-tool support for developing these methodologies, the definition of the method base should conform to a standard formalism. To address this requirement, we have adopted the Eclipse Process Framework Composer (EPFC) [10], which is an open-source situational method engineering tool platform. EPFC provides an extensible platform for assembly-based method engineering, and is fully extensible through provision of facilities for adding new method plug-ins, method packages, and libraries. Since EPFC conforms to the OMG's Software and System Process Engineering Metamodel (SPEM 2.0) [11] in the decomposition of software processes into their building blocks, we have thus adopted SPEM 2.0 for the definition of the proposed agile method base. In SPEM 2.0, processes are composed of standard reusable components that constitute the method content. The method content is composed of elements of three types: roles, work products, and tasks. In order to define the lifecycle of a process, illustrate different levels of abstraction, and constrain the order in which activities are performed, SPEM 2.0 incorporates the notions of lifecycle, phase, activity, task and technique – in descending order of granularity. The same formalism has been followed for process decomposition in our proposed agile method base.

Having delineated the outline of our approach, the rest of this paper is organized as follows: Section 2 briefly reviews the literature related to this work and highlights the contributions of this research; Section 3 covers the first step in defining the agile method base by delineating the methodology requirements of an agile development process; Section 4 completes the specification of the agile method base by introducing agile method chunks and relating them to the methodology requirements that they address; and Section 5 presents the concluding remarks and discusses possible directions for furthering this research.

II. Related Research

Research efforts focusing on providing CAME-tool support for agile methodologies can be categorized in the following three classes, according to the assembly-based SME stage that they address:

- Research efforts which either explicitly focus on eliciting requirements for software development methodologies, or implicitly address methodology requirements by proposing evaluation criteria for methodologies [12, 13, 14].
- Research efforts which address the application of situational method engineering to the development of agile methodologies in the following three areas:
  a. Introduction and adoption of agile practices, processes and activities; from among these, we have scrutinized the most prominent ones, including DSDM [2], Scrum [3], XP [4], ASD [5], Crystal Clear [6] and FDD [7].
  b. Extraction and integration of agile best practices and activities into unified frameworks, metamodels, and process patterns; the methods proposed in [15, 16, 17] are good examples.
  c. Enhancement of existing agile processes with successful practices and techniques; among these are the research efforts reported in [18, 19, 20, 21, 22, 23].
- And finally, research efforts which explore the implementation of agile methodologies in CAME tools, for which the only resource that we have found is the EPFC platform, already providing support for instantiation of XP and Scrum processes [10]. However, this support is partial, since it is limited to the proper instantiation of just two agile processes, and does not provide assembly-based method engineering support in this context.

The novelty of our approach is in the combination of SME steps. This research is distinguished from previous research endeavors in two aspects:

- It adopts the systematic approach of assembly-based method engineering in its entirety, and applies it in the context of agile methodologies.
- It takes the most important step towards enhancing CAME-tool support for situational engineering of agile method by covering the first three steps of assembly-based method engineering, and proposing an agile method base based on SPEM 2.0 formalisms. Conforming to SPEM 2.0 results in a standard process decomposition scheme which can be immediately plugged into CAME tools supporting this standard. Moreover, it provides an initial standardized basis which can be further extended and enriched by defining new method chunks.

III. Agile Methodology Requirements

The situation-specific nature of methodology features and requirements necessitates the application of method engineering practices. Each project situation distinguishes itself from others by a set of characteristics which should be addressed by the methodology constructed for it. Thus, in the process of situational method engineering, the characteristics of the project situation become a set of requirements for the method under construction. Accordingly, in the assembly-based strategy, method chunks are picked out from the method repository based on the requirements that they should satisfy. Consequently, the first step in the design of an agile method base is to identify a generic, comprehensive set

of possible methodology requirements to be satisfied by the proposed method chunks. In this section, we introduce a set of fundamental requirements which arise in the context of agile software development. The main factor in the elicitation of the set of method requirements proposed herein is the consequences which result from the natural compliance of agile methods to the principles of agility [1].

Since software development processes are composed of two basic types of activities, namely project management activities and software development activities, we have categorized the method requirements into the two groups of management requirements and development requirements, as delineated throughout the rest of this section:

a) ***Management Requirements:***

- *Project Management*: Deviation from plan-driven approaches, in which plans and schedules are firmly adhered to, and welcoming change even in the late phases of development, puts a strong demand on project management issues in agile methodologies. Therefore, project management requirements are of paramount importance to the success of an agile methodology.
- *Risk Management*: Risks which are characterized with the three factors of uncertainty, loss (lack of required knowledge, time, or resources), and finite duration, can adversely affect the progress of the development process if they are not handled appropriately at the process level. Although the iterative-incremental nature of agile methodologies satisfies this requirement to some extent, the ever-changing environment of these methodologies means that risks need to be handled more elaborately. This poses risk management as a challenging requirement in agile methodologies.
- *Quality Assurance*: The emphatic attention of agile methodologies to technical excellence and good design is somewhat eroded by their emphasis on "maximizing the amount of work not done". Quality assurance is therefore a challenging requirement in agile methodologies.
- *Configuration Management*: Frequent delivery of an executable product in a short period of time, in addition to frequent application of change in the delivered product, demonstrates the inevitable importance of configuration management activities in agile methodologies.

b) ***Development Method Requirements***

- *Complexity Management*: Lightweight agile processes should be capable of accommodating the complexities arising in their ever-changing environments. Hence, complexity management should be handled properly in agile methodologies.
- *Basis in Requirements*: Strict adherence to customer requirements requires stringent strategies to be adopted by agile methodologies.
- *Traceability to Requirements*: The frequent changes in requirements faced during the lifecycle of agile methodologies, along with the methodology requirement "basis in the requirements", raises the need for tracing a product to the requirements that it addresses.
- *Seamlessness*: Seams arise when untraceable changes are introduced during transitions between different phases of methodologies, and in the transformation of the outputs of one phase to the inputs of another phase. Seams give rise to unresolved faults and defects, thus violating the method requirement of "traceability to requirements". The agile practice of pruning input and output products, and replacing them with face-to-face conversation, contradicts seamlessness. Thus, preserving this feature has become an essential requirement in agile methodologies.
- *Flexibility*: Due to their turbulent and risky environments and the frequent changes occurring in their process frameworks, agile methodologies should be equipped with adjustment and tuning activities in order to remain effective.
- *Maintainability*: The high frequency of product delivery, and the rate at which the requirements and the product change in agile methodologies, necessitates special attention to maintainability issues.
- *Usability*: A methodology should deliver the products which fit best to their usage context. The need for this feature is intensified in agile methodologies, which put great emphasis on customer satisfaction by adopting it as a fundamental principle.
- *Support for Distributed teams*: Face-to-face conversation, as recommended in the context of agility, is in conflict with modern outsourcing policies, in which a project is developed across geographically dispersed locations.

IV. THE PROPOSED AGILE METHOD BASE

In this section, we propose an agile method repository composed of a set of agile method chunks. In the extraction of the proposed method chunks, we have extensively scrutinized the prominent processes in the context of agile development, including DSDM, Scrum, XP, ASD, Crystal Clear, and FDD, along with their enhancements and augmented practices and activities, as proposed in [18, 19, 20, 21]. The most common workflows taking place during the lifecycle of an agile methodology are delineated in this section.

Although there are several research efforts similar to ours in presenting an all-embracing paradigm for agile methodologies – such as ASDLC (Agile System Development Life Cycle) [15], FRAME (FRactal Agile MEtamodel) [16] and process patterns for agile methodologies [17] – this research is different in two aspects:

1) Application of a method engineering approach and specification of agile workflows based on process elements conforming to the SPEM 2.0 metamodel. Regarding this, each workflow is depicted based on the roles taking part in that workflow, the tasks performed by the roles, and the work products which are input and output to each task. Previous research has been solely focused on the activities involved.

2) Decomposing the overall lifecycle of agile methodologies into different levels of granularity –

following SPEM2.0 metamodel – including *phases*, *activities*, *tasks* and *techniques,* in descending order of granularity. Previous research efforts neglect some of these different levels of granularity.

The above explanations specify the overall structure of our proposed method base and its method chunks. To delineate the constituents of the method base, we have first identified the relevant roles participating in agile workflows – along with their responsibilities – in section A; we have then demonstrated the work products developed in agile workflows in section B; subsequently, a description of the tasks performed during the lifecycle of agile methodologies is provided in section C. The delineation of each task is accompanied by the source process from which it has been extracted.

Finally, to provide an overall overview of our proposed method base, we have summarized the decomposition of the lifecycle of agile methodologies into the specified method chunks – along with the method requirements that they address – in Table II.

A. Roles

The typical roles participating in agile workflows – along with their responsibilities – have been briefly explained below:

- **Customer**: A user representative in charge of interacting with the development teams in order to help in eliciting system requirements, prioritizing requirements, writing functional tests, and providing feedback on the delivered product.
- **Product owner**: An individual from the user community who is in charge of prioritizing and maintaining the requirements, and keeping track of their business values and their interrelationships. This role is also responsible for the success and failure of the project.
- **Domain expert**: A customer, business analyst, or product owner who provides the development teams with proper knowledge about user requirements.
- **User-product owner**: An individual from the development team (first proposed in [18]) who is in charge of establishing communication between customers, product owner and domain experts to provide them with a proper vision of the system to be developed.
- **Programmer**: An individual in charge of designing, coding and testing the delivered product.
- **Chief programmer**: Expert developer(s) in charge of managing the project, forming the development team(s), assigning roles to individuals, and monitoring development activities.
- **Release manager**: An individual in charge of applying software configuration management activities during the frequent deliveries of the system under development.
- **Modeling team**: A team of programmers, chief programmers and domain experts in charge of modeling and designing the system under development.
- **Tester/Analyst**: An individual in charge of documenting the requirements specification, testing the system, and keeping track of the requirements and their relevant tests.
- **Technical coordinator**: An individual who is in charge of developing the system architecture, and controlling and assuring the technical quality features of the project and its products.
- **Technical writer**: An individual who is in charge of documenting the user manuals and guidelines.
- **Tracker**: An individual in charge of keeping track of the project progress and the velocity of task accomplishments, and providing feedback and estimations on project tasks.
- **Development team**: A team consisting of programmers, chief programmer(s), tester/analysts, technical coordinator(s), technical writers, the release manager, and tracker.
- **Project community**: A community of all the people involved or affected by the project, including the development team(s), customer, product owner, financial and executive sponsors, and marketing team(s).

B. Work Products

To present the work products produced during the lifecycle of agile methodologies, we have categorized them into three standard classes of *Artifacts*, *Outcomes* and *Deliverables,* in accordance with SPEM 2.0. A brief definition of each category is provided below:

- *Artifact*: A tangible work product that is consumed, produced, or modified by one or more tasks. Roles use artifacts to perform tasks and to produce other artifacts.
- *Outcome*: An intangible work product that may be a result or state. Outcomes may also be used to describe work products that are not formally defined.
- *Deliverable*: A collection of work products, usually artifacts. Deliverables are also used to represent an output from a process that has material value to a client, customer, or other stakeholder.

Based on the definition of each class, we have categorized the work products of our method base in Table I.

C. Tasks

To present the tasks taking place during the lifecycle of agile methodologies, we have organized them into four separate phases of: *Project Initiation*, *Development*, *Review* and *Project Termination*. These phases are accordingly broken down into a set of activities, namely *Project Start-up*, *Initial Exploration*, *Requirements Elicitation*, *Release Planning*, *Iteration Planning*, *Build*, *Reflection* and *Project Wrap-up*.

The outlines of the phases, their relevant activities, and the interrelationships are illustrated in Fig. 1. Each activity is in turn composed of a set of tasks which should be performed in order to accomplish the relevant activity.

Throughout the rest of this section, we delineate these tasks and their related techniques, the roles involved, and the resulting work products, ordered by their relevant phases. The outline for each of the four phases is depicted in Fig. 2, Fig. 3, Fig. 4, and Fig. 5, respectively.

TABLE I. AGILE WORK PRODUCTS

Type	Work products
Outcome	Feasibility report
	Initial methodology
	Project teams report
	Model notes
	Project velocity
	Release plan
	Roles specification
	Iteration plan
	Test notes
	Refined methodology
	Reformed teams
	Recalibrated release plan
	Validation report
Artifact	User Experience (UE) vision
	Overall object model
	High-level requirements
	Prioritized requirements
	Requirements specification
	High-level architecture
	Design notes
	Design package
	User manuals
	Postmortem report
Deliverable	Iteration release
	Final product

1. Project Initiation Phase

- **Analyze Feasibility** (DSDM, Crystal Clear): This task, performed by the project community, deals with assessing the feasibility and applicability of the methods to be developed under the specific circumstances of a project. The output of this task is the feasibility report, which determines the next step to be taken.

***Related techniques*:** *Apply suitability filter* (DSDM)- This technique checks the suitability of a method for the project at hand by applying a list of project- and organizational criteria.

- **Build the development teams** (Scrum, XP, Crystal Clear): In this task, the structure of the teams involved in a development process is determined by the project community. This results in the identification of issues such as the size of the teams and the distribution of skills among the teams. The output of this task is a report on team structure which is updated in subsequent phases.

***Related techniques*:** *Develop Self-organized teams* (Scrum)- In this technique, development teams of diversely skilled members are formed without any centralized control.

- **Shape the methodology** (Crystal Clear): In this task, the skeleton of a methodology is shaped by identifying an initial set of rules and working conventions which should be followed by the teams. This set will be continuously revised at the end of each execution of the development phase.

- **Elicit user experience** ([18])**:** In this task, in order to obtain a thorough understanding of the user environment and user requirements, a dedicated role – User-product owner – is assigned in order to get in touch with the user community and provide an updated version of the user vision identifying the exact delineation of the users' expectations from the system under development.

- **Conduct domain walkthrough** (FDD): In this task, an overview of the problem domain is presented by the domain experts. This outputs a high-level description of the problem domain identifying the outline of the system to be developed, including the overall functional requirements, data models, and guidelines.

- **Develop the object model** (FDD): Based on the scope of the problem domain specified in the high-level description of the system, modeling team(s) develop an overall object model of the system which identifies the overall structure (objects), functions (methods) and behavior (the sequence of method calls) of the system to be developed. Attached to the object model are model notes which provide detailed explanation for each of its parts.

Related techniques: *Develop Small Group model* (FDD)- To assure the quality of the models developed and to manage their complexity, the problem domain is partitioned into separate areas and each area is examined by several individual modeling teams. This outputs several models for each area, one of which (or a combination) is approved to be integrated into the overall object model of the system.

- **Generate product backlog** (Scrum, XP, Crystal Clear): In this task, the high-level requirements identified in the user vision are inserted into a repository of requirements called the product backlog – a term borrowed from Scrum– to be managed and updated by the product owner. This provides the development team(s) with an overall insight into the amount of work to be done.

- **Prioritize the product backlog** (Scrum, XP, Crystal Clear): In this task, product backlog items are prioritized by the product owner according to their business value and the risks involved in their development.

Related techniques: *Apply MoSCoW rules* (DSDM)- In this technique, the requirements are prioritized based on their importance into four groups of Must-have, Should-have, Could-have and Won't-have features.

- **Document the requirements** ([20, 22]): In this task, the information on project requirements, which is mostly exchanged through face-to-face communication, is documented by tester/analysts.

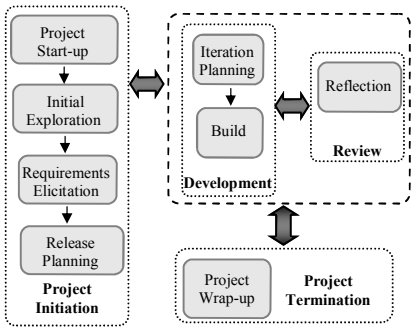

Figure 1. Overall phases and activities of agile methodologies

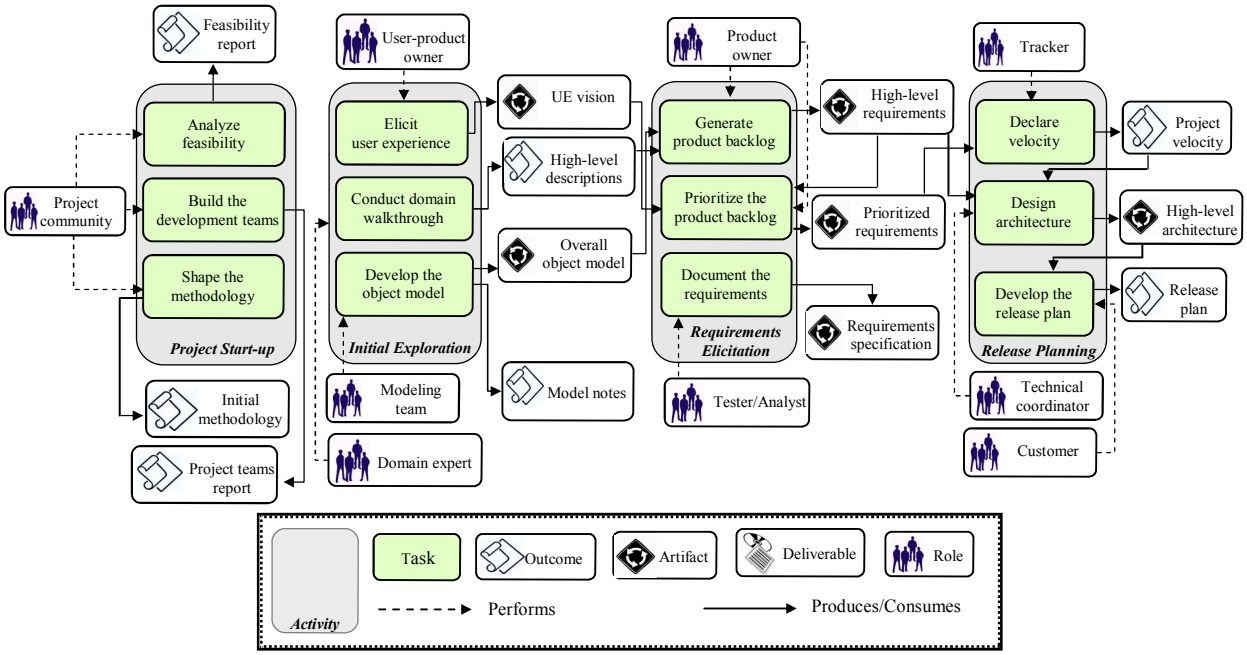

Figure 2. Workflows of the Project Initiation phase

- **Declare velocity** (XP): In this task, based on the prioritized items of the product backlog, the velocity of the project, which is the expected amount of work to be done in each execution of the development cycle, is specified by the tracker. This results in the declaration of project velocity.

Related techniques: *Use Burn-chart* (Crystal Clear)- In this technique, to monitor the progress of a project, a special chart is developed with time on its x-axis and the work-units (use cases or modules) on its y-axis. This chart demonstrates the overall progress of the project.

- **Develop the release plan** (Scrum, XP, FDD): in this task, based on the velocity declared, a list of requirements and goals of the highest priority are specified by the customer and product owner to be achieved in the next execution of the development phase. This results in the delineation of the iteration scope.

- **Design architecture** (Scrum, XP, ASD, Crystal Clear): In this task, an initial, abstract schema of the system and its components is sketched by the technical coordinator, to be refined continuously in subsequent phases.

***Related techniques*:** *Create system metaphor* (XP)- In this technique, a high-level story of the system narrating the system's functionality and architecture is provided which provides the common vision and vocabulary shared between customers and developers.

Explore technology alternatives (Crystal Clear)- In this technique, different technologies which could be applied to each component of the system are examined, and the most promising one is adopted.

Hold Joint Application Design (JAD) session (DSDM, ASD)- In this technique, in order to sketch the overall architecture of the system under development and its key features, and to specify the different components of the system, a meeting is held by the project community in which all the stakeholders discuss the different aspects of the system.

2. Development Phase

- **Break down into tasks** (Scrum, XP, Crystal Clear, FDD): In this task, the iteration scope is broken down into concrete programming tasks which require roughly the same amount of time to be developed. This task is performed by the chief programmer.

Related techniques: *Divide based on functionality* (FDD)- In this technique, the system is divided into subsystems based on its structure and functionality.

Divide based on skills and time base (DSDM)- In this technique, tasks are broken down based on the amount of effort and the skills required for developing them.

- **Evaluate velocity** (XP, Crystal Clear): In this task, the velocity of previous executions of the development cycle is evaluated by the tracker, and the result is used in order to adjust the velocity of the current development phase.

- **Develop rough estimates** (Scrum, XP): In this task, an estimation of the development time and effort is assigned by the programmers to fine-grained tasks. The estimates become more accurate as the project velocity is reassessed after each execution of the development phase.

***Related techniques*:** *Prototype-based planning* (DSDM, Scrum)- In this technique, in order to resolve ambiguities on the amount of time and effort required, quick throwaway prototypes of the system are developed.

Card-based/Blitz planning (XP, Crystal Clear)- In this technique, tasks are written down on separate cards such that there is no task dependency between the different cards.

Programmers then estimate the time and effort required to develop each task.

- **(Re) Assign responsibilities** (Scrum, XP, Crystal Clear, FDD): In this task, based on the rough estimates specified, a set of tasks is assigned/reassigned to each programmer in each execution of the development phase. The tasks are assigned so that the workload of the team members stays equal.

***Related techniques*:** *Task sign-up* (Scrum, XP)- In this technique, programmers voluntarily choose a set of tasks to complete during an execution of the development cycle.

Assign to class owners (FDD)- In this technique, the chief programmer assigns each set of the tasks (features) to each individual programmer.

- **Hold Daily stand-up meeting** (Scrum, XP, Crystal Clear): During the development phase, a daily meeting is held between the development-team members to distribute information, communicate problems and solutions, and help the team stay on track.

- **Hold design session** (XP, FDD): In this task, a meeting is held in which the chief programmers and/or programmers design the parts of the system which should be developed in the current development phase. However, the weight of this task, its importance and its outputs differ according to the situation at hand.

Related techniques: *Conduct design inspections* (FDD)- In this technique, team effort is focused on the design task. By involving the programmers, different parts of the system are designed carefully by inspecting and updating the overall object model produced in the Project Initiation phase, investigating different design alternatives, and developing detailed sequence diagrams. The result of this technique is a design package consisting of the detailed design of the system (classes and their attributes and functions).

- *Hold quick design session* (XP)- In this technique, a 30-minute design session is held among the chief programmers, in which the system part under development is designed. The output of this meeting is CRC cards indicating the classes of the system along with the required UML diagrams (typically consisting of activity-, collaboration-, and sequence diagrams).

- **Code** (DSDM, XP, Scrum, ASD, Crystal Clear, FDD): In this task, executable parts of the system under development are produced by programmers.

Related techniques: *Collective code ownership* (XP)- In this technique, in order to accelerate the development, all the code is put in a shared repository, so that all programmers have access to it in order to add functionality or fix bugs.

Apply stringent coding standard (XP)- In order to enhance communication among programmers, a common coding standard is enforced, mainly to promote code legibility.

Pair programming (XP, ASD)- In this technique, programmers work in pairs, with each pair working on the same machine and the same code.

Side-by-side programming (Crystal Clear)- In this technique, programmers work in pairs. While they work on different programming tasks, they help each other with the problems encountered.

One owner per class (FDD)- In this technique, the development of each class, module or component is assigned to a *class owners*, who is solely responsible for creating and updating the parts thus assigned.

- **Test** (Scrum, XP, Crystal Clear): In this task, the code developed is tested by testers/analysts in order to identify and remove the bugs.

Related techniques: *Test-driven development* (Scrum, XP, Crystal Clear)- In this technique, tests are produced before the coding task begins. Therefore, test cases are developed based on the requirements, and code fragments are then produced to satisfy these test cases.

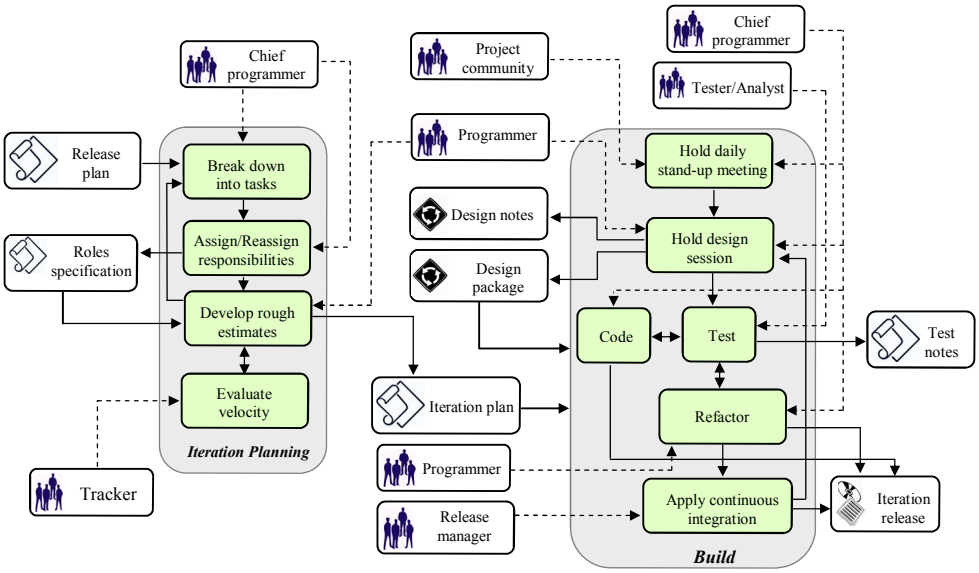

Figure 3. Workflows of the Development phase

- **Refactor** (Scrum, XP, Crystal Clear, FDD): In order to eliminate redundancy, simplify and refine the code without changing its external behavior, and enhance code readability, refactoring should be done continuously on the code developed.

- **Apply continuous integration** (Scrum, XP, Crystal Clear): Due to frequent builds of software, the integration task is of main concern in agile methods, and should be applied continuously.

3. Review Phase

- **Fine-tune the methodology** (ASD, Crystal Clear): After each release of the product, the methods used are revised and refined in order to remove the weaknesses of the methods and reinforce their strengths. To this aim, the problems of the method used are identified and removed, and new conventions are added to the existing ones.

Related techniques: *Hold reflection workshop* (Crystal Clear)- In this technique, after each release of the system, a workshop is held in which the project community reviews and tunes the processes in use.

- **Recalibrate the release plan** (DSDM, Scrum, XP, ASD, Crystal clear): After each release, the experience gained of the project velocity, the achievement of the iteration scope and the estimates made about the tasks during the development, are reflected to the release plan. The release plan is thus updated and refined iteratively.

- **Tune the development teams** (XP, Crystal Clear, ASD): In this task, the progress of the development teams is evaluated and team structures are tuned in order to increase their performance, remove workload bottlenecks, and distribute development information among the teams and individuals.

Related techniques: *Move people around* (XP, Crystal Clear, ASD)- In this task, parallel teams are introduced, and team members are moved around in order to bring the project to its desirable status, remove bottlenecks, and distribute information among the teams. Proper flow of knowledge and experience is thus ensured.

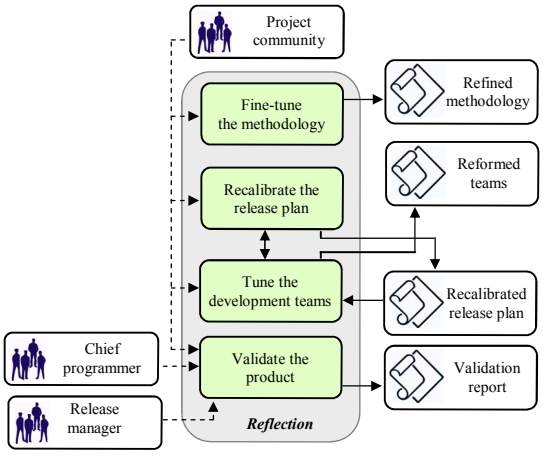

Figure 4. Workflows of the Review phase

- **Validate the product** (DSDM, ASD): In this task, the released product goes through acceptance tests by the customers and the release manager to validate the functionality of the system produced so far. This results in adding new requirements to the product backlog, and changing or reversing the implemented requirements in case faults or defects are detected in the system. Based on the result of this task, which is documented in the validation report, either another execution of the development phase is commenced, or the project enters the termination phase.

4. Project Termination Phase

- **Prepare the environment** (Scrum, XP, DSDM, Crystal Clear): In order to deliver the final product to the customer, all the necessary resources for the final system are installed, and system manuals and documents are prepared by the technical writer.

- **Release the final product** (DSDM, XP, Scrum, ASD, Crystal Clear, FDD): In this task, the final version of the system is released and integrated with other components and systems.

- **Apply system-wide testing** (Scrum, XP, Crystal Clear): Having released the final system, in this task, system-wide tests are applied by the project community and customer to ensure the correct operation of the system. If no errors are reported, deployment will ensue.

- **Deploy** (DSDM, Scrum, XP, Crystal Clear): In this task, the system is deployed into the user environment, necessary system conversions are carried out, system users and operators are trained, and final tuning and stabilization activities are conducted.

- **Conduct postmortem activities** (XP, Crystal Clear): This task, performed by the chief programmers and relevant members of the project community, involves compiling and recording the lessons learned from the project, in order to be used in future projects. This results in the postmortem report, which reflects the experience gained during the project by all the people involved.

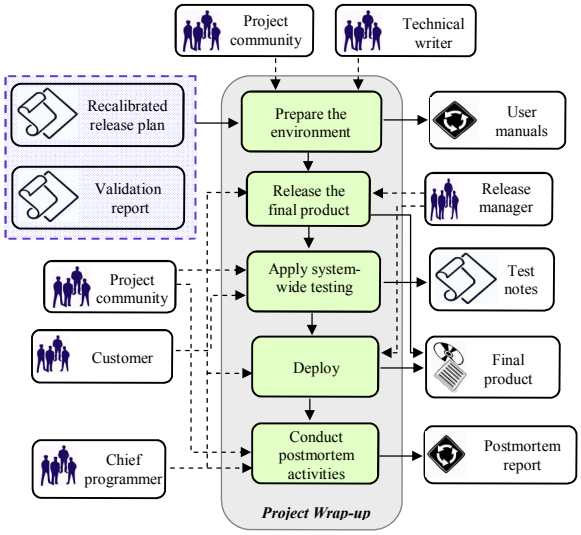

Figure 5. Workflows of the Project Termination phase

TABLE II. OVERVIEW OF AGILE METHOD CHUNKS AND THE REQUIREMENTS ADDRESSED

Phase (Activity)		Tasks	Techniques	Related roles	Work-products	Method Requirements Addressed
Project initiation	Project Start-up	Analyze feasibility	Apply suitability filter	Project community	Feasibility report	Risk management
		Shape the methodology	---	Project community	Initial methodology	Risk management, Flexibility, Complexity management
		Build the development teams	Develop self-organized teams	Project community	Project teams report	Distributed teams, Project management
	Initial Exploration	Elicit user experience	---	User-product owner	User experience vision	Risk management, Usability
		Conduct domain walkthrough	---	Domain expert	High-level descriptions	Risk management
		Develop the object model	Develop small group model	Modeling team	Overall object model, Model notes	Complexity management, Risk management, Seamlessness
	Requirements Elicitation	Generate product backlog	---	Product owner	High-level requirements	Quality assurance, Basis in requirements, Traceability to requirements
		Prioritize the product backlog	Apply MoSCoW rules	Product owner	Prioritized requirements	Risk management, Project management
		Document the requirements	---	Tester/Analyst	Requirements specification	Maintainability, Complexity management, Risk management
	Release Planning	Declare velocity	Use Burn-chart	Tracker	Project velocity	Risk management, Project management, Complexity management
		Design architecture	Create system metaphor, Explore technology alternatives, Hold JAD session	Technical coordinator	High-level architecture	Risk management, Complexity management, Project management, Quality assurance
		Develop the release plan	---	Customer, Product owner	Release plan	Project management
Development	Iteration Planning	Break down into tasks	Divide based on functionality, Divide based on skills and time base	Chief programmer	---	Project management, Risk management
		Assign/Reassign responsibilities	Task sign-up, Assign to class owners	Chief programmer	Roles specification	Project management, Risk management, Quality assurance
		Develop rough estimates	Prototype-based planning, Card-based/Blitz planning	Programmer	Iteration plan	Project management, Risk management, Configuration management
		Evaluate velocity	---	Tracker	---	Project management, Risk management
	Build	Hold daily stand-up meeting	---	Project community, Chief programmer	---	Project management, Risk management, Quality assurance
		Hold design sessions	Conduct design inspections, Hold quick design session	Chief programmer, Programmer	Design notes, Design package	Risk management, Project management, Complexity management, Seamlessness
		Test	Test-driven development	Tester/Analyst	Test notes	Quality assurance, Risk management, Project management
		Code	Collective code ownership, Pair programming, Side-by-side programming, Apply stringent coding standard, One owner per class	Chief programmer	Iteration release	Quality assurance, Risk management, Maintainability, Project management, Complexity management
		Refactor	---	Chief programmer, Programmer	Iteration release	Maintainability, Quality assurance
		Apply continuous integration	---	Release manager	Iteration release	Risk management, Quality assurance
Review	Reflection	Fine-tune the methodology	Hold reflection workshop	Project community	Refined methodology	Risk management, Complexity management, Flexibility
		Recalibrate the release plan	---	Project community	Recalibrated release plan	Project management, Risk management
		Tune development teams	Move people around	Project community	Reformed teams	Maintainability, Risk management, Project management
		Validate the product	---	Release manager, Chief programmer	Validation report	Quality assurance, Risk management, Project management, Usability
Project Termination	Project Wrap-up	Prepare the environment	---	Project community, Technical writer	User manuals	Risk management, Project management
		Release the final product	---	Release manager, Chief programmer	Final product	---
		Apply system-wide testing	---	Project community, Release manager, Customer	Test notes	Risk management, Quality assurance
		Deploy	---	Release manager, Chief programmer	Final product	---
		Conduct postmortem activities	---	Project community, Chief programmer	Postmortem report	Risk management, Quality assurance

V. CONCLUSIONS AND FUTURE WORK

In this research, the ultimate objective is to apply the assembly-based SME approach to the development of bespoke agile methodologies. To this aim, as the first step, we have extensively explored successful agile processes, practices and techniques, and have proposed an agile method base. In the development of the proposed method chunks, SPEM-2.0 conventions have been followed; the processes have therefore been decomposed into three types of elements: tasks, roles, and work products. This enhances the usability of the proposed method base in CAME tools that conform to the SPEM-2.0 metamodel, including the EPFC tool.

As the next step, a process will be defined for assembling the method chunks of the proposed method base into a coherent methodology. This includes the provision of guidelines and configuration rules for the assembly process. These steps together provide a comprehensive method framework for assembly-based engineering of agile methodologies. The final result will be integrated into the EPFC tool. The validity of the integrated framework will be examined in the context of a concrete SME project: the specific requirements of a realistic project situation will be elicited, and a complete situation-specific methodology which satisfies these requirements will be developed through applying the enriched version of the EPFC tool.

ACKNOWLEDGEMENT

We wish to thank Iran Telecommunications Research Center (ITRC) for sponsoring this research.

REFERENCES

[1] K. Beck et al., "Manifesto for agile software development," published on the web at: http://agilemanifesto.org.

[2] J. Stapleton, "DSDM: Business focused development," 2nd edition, Addison Wesley, 2003.

[3] K. Schwaber and M. Beedle, "Agile software development with Scrum," Prentice-Hall, 2001.

[4] K. Beck and C. Andres, "Extreme programming explained: Embrace change," 2nd edition, Addison Wesley, 2004.

[5] J. Highsmith, "Adaptive software development: A collaborative approach to managing complex systems," Dorset House, 2000.

[6] A. Cockburn, "Crystal clear: A human-powered methodology for small teams," Addison Wesley, 2004.

[7] S. R. Palmer and J. M. Felsing, "A practical guide to feature-driven development," Prentice-Hall, 2002.

[8] J. Ralyté, S. Brinkkemper and B. Henderson-Sellers, "Situational method engineering: Fundamentals and experiences," Springer, 2007.

[9] A. Niknafs and R. Ramsin, "Computer-aided method engineering: An analysis of existing environments," Proc. 20th International Conference on Advanced Information Systems Engineering (CAiSE'08), LNCS 5074, Springer, 2008, pp. 525-540, doi: 10.1007/978-3-540-69534-9_39.

[10] P. Haumer, "Eclipse Process Framework Composer," Eclipse Foundation, 2007.

[11] R. Bendraou, B. Combemale, X. Cregut and M. Gervais, "Definition of an executable SPEM 2.0," Proc. 14th Asia-Pacific Software Engineering Conference (APSEC'07), December. 2007, pp. 390-397, doi: 10.1109/ASPEC.2007.60.

[12] R. Ramsin and R. F. Paige, "Process-centered review of object oriented software development methodologies," ACM Computing Surveys, vol. 40, no. 1, February. 2008, pp. 1-89, doi: 10.1145/1322432.1322435.

[13] J. Koskela, "Software configuration management in agile methods," VTT Publications, 2003.

[14] M. Taromirad and R. Ramsin, "CEFAM: Comprehensive Evaluation Framework for Agile Methodologies," Proc. 32nd Annual IEEE Software Engineering Workshop (SEW'08), December. 2008, pp. 195-204, doi: 10.1109/SEW.2008.19.

[15] S. W. Ambler, "The agile system development lifecycle," published on the web at: http://www.ambysoft.com/essays/agileLifecycle.html.

[16] M. Hasani Sadi and R. Ramsin, "FRAME: A generic fractal process metamodel for agile methodologies," Proc. 7th International Conference on Software Engineering Research, Management and Applications (SERA'09), December. 2009, pp. 251-264, doi: 10.1007/978-3-642-05441-9_22.

[17] S. Tasharofi and R. Ramsin, "Process patterns for Agile methodologies," In Situational Method Engineering: Fundamentals and Experiences, J. Ralyté, S. Brinkkemper and B. Henderson-Sellers (Eds.), Springer, 2007, pp. 222-237, doi: 0.1007/978-0-387-73947-2.

[18] M. Singh, "U-Scrum: An agile methodology for promoting usability," Proc. Agile 2008 Conference, IEEE Press, August. 2008, pp. 555-560, doi: 10.1109/Agile.2008.33.

[19] K. Sureshchandra and J. Shirinivasavadhani, "Adopting agile in distributed development," Proc. IEEE International Conference on Global Software Engineering (ICGSE'08), August. 2008, pp. 217-221, doi: 10.1109/ICGSE.2008.25.

[20] J. Nawrocki, M. Jasiński, B. Walter and A. Wojciechowski, "Extreme programming modified: Embrace requirements engineering practices," Proc. 10th Joint IEEE International Requirements Engineering Conference (RE'02), September. 2002, pp. 303-310, doi: 10.1109/ICRE.2002.1048543.

[21] J. Dajda and G. Dobrowolski, "Experiment-driven approach to building support for distributed agile teams," Proc. 14th Asia-Pacific Software Engineering Conference (APSEC'07), December. 2007, pp. 398-405, doi: 10.1109/ASPEC.2007.12.

[22] T. Karamat and A. N. Jamil, "Reducing test cost and improving documentation in TDD (Test Driven Development)," Proc. Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06), IEEE Press, June. 2006, pp. 73-76, doi: 10.1109/SNPD-SAWN.2006.59.

[23] M. Huo, J. Verner, L. Zhu and M. A. Babar, "Software quality and agile methods," Proc. 28th Annual International Computer Software and Applications Conference (COMPSAC'04), IEEE Press, September. 2004, pp. 520-525, doi: 10.1109/CMPSAC.2004.1342889.

[24] T. Dingsøyr and G. K. Hanssen, "Extending agile methods: Postmortem reviews as extended feedback," Proc. 4th International Workshop on Learning Software Organizations, LNCS 2640, Springer, 2002, pp. 4-12, doi: 10.1007/b94220.

[25] L. Crispin and T. House, "Testing extreme programming," Addison Wesley, 2002.

[26] W. Wake, "Extreme programming explored," Addison Wesley, 2001.

[27] G. Succi, and M. Marchesi, "Extreme programming examined," Addison Wesley, 2001.

[28] R. Jeffries, A. Anderson and C. Hendrickson, "Extreme programming installed," Addison Wesley, 2001.