FRAME: A Generic Fractal Process Metamodel for Agile Methodologies

Mahsa Hasani Sadi and Raman Ramsin

Department of Computer Engineering, Sharif University of Technology
Azadi Avenue, Tehran, Iran
mhsadi@ce.sharif.edu,ramsin@sharif.edu

Summary. The widespread need for flexibility and adaptability in software development processes has resulted in the emergence of Agile practices and principles. Although different in fine-grained detail, all agile methodologies follow a common approach in their processes. To expose the common paradigm behind the design of agile processes, we have developed FRAME (FRactal Agile MEtamodel), a generic fractal process metamodel for agile process frameworks. Recursive application of FRAME at different levels of process design results in the specification of a configurable process framework; this framework serves as a basis for constructing agile methodologies through paradigm-based Situational Method Engineering (SME). In order to explore how FRAME is manifest in existing agile processes, relevant activity patterns have been extracted from seven prominent agile processes, thus forming a repository of concrete FRAME components.

1 Introduction

The high degree of flexibility and relatively low cost of change in software products has shifted the focus of software development methodologies from product-centered approaches to process-centered ones. This was due to the need for processes that can accommodate adaptability and sustain the flexibility of products. The turbulence existent in the target environments of software systems amplified this need. To address these issues, agile methodologies emerged as a substitute for heavyweight methodologies such as RUP and Catalysis [1]. In contrast to heavyweight methodologies, which put great emphasis on detailed specification of processes as steps in the gradual refinement of products, agile methodologies focus on process frameworks which provide adaptability. Hence, agile methodologies are mainly configured with activities and patterns of adaptable frameworks, leaving the fine-grained steps of product evolution as a degree of freedom, to be specified and tuned to fit the situation at hand. This has turned agile methodologies into configurable process frameworks rather than concrete processes. The general outline of these configurable frameworks has been specified in the Agile Manifesto [2] and the Agile Principles [3]. To realize these specifications in their processes, agile methodologies are instantiated with a wide range of practices and activity patterns, all aiming at implementing adaptability at the process level. The diversity of these practices and patterns has resulted in the emergence of a wide range of agile methodologies; some of these processes provide specific guidelines for configuring their constituent components, while others are just

R. Lee & N. Ishii (Eds.): Soft. Eng. Research, Manage. & Appli. 2009, SCI 253, pp. 251–264.
springerlink.com

a result of scavenging and merging features from other methodologies. This diversity has left the definition of a generic agile framework as an open problem.

Much work has been done to present a unified front for these efforts, most of which exploit Situational Method Engineering (SME) approaches [4]. Different strategies in SME have been explored in this context, resulting in different solutions. The adoption of *assembly-based* approaches has resulted in the extraction of method-chunks and reusable frameworks from prominent agile methodologies, to be reused independently for instantiating agile processes. Instances of this approach can even be found in prominent methodologies; examples include the enhancement of the SCRUM [5] abstract framework with practices and patterns from XP [6], thus converging these two methodologies toward a unified process. Adoption of *extension-based* approaches can be observed in the definition of the FDD [7] methodology, the process of which is a pruned minimal core that can be extended into a full methodology through applying augmentation patterns. The applicability of *paradigm-based* approaches – aiming at providing process metamodels as a generic substrate based on which agile methodologies can be instantiated – has also been explored in the context of agility: This can be seen in the generic Agile System Development Life Cycle (ASDLC) [8], which outlines the general schema of phases and core activities applied in agile methodologies. This effort has been complemented by a set of agile process patterns, which specify the overall phases, stages, and tasks seen prominent agile methodologies [9].

Among the various research areas explored, the application of paradigm-based approaches to engineering agile methodologies is particularly significant. This is due to the potential of such approaches for providing an infrastructure for developing Computer-Aided Method Engineering (CAME) tools [10]. For instance, the Eclipse Process Framework Composer (EPFC), which is based on the SPEM-2 metamodel [11], is a CAME tool which can support the development of agile methodologies. Efforts conducted so far in different strands of method engineering – though contributing much to the definition of a framework for agile development – are restricted to the extraction and reuse of method-chunks and patterns from existent agile methodologies and their integration into a generic process template.

With the focus on paradigm-based engineering of agile methodologies, we propose FRAME (FRactal Agile MEtamodel) as a generic fractal process metamodel for agile methodologies. FRAME distinguishes itself from approaches previously explored by presenting a generic fractal model based on the general approach of adaptive process frameworks [12, 14, 15]; it therefore does not limit itself to a specific set of agile processes. The idea of adaptive process frameworks was introduced in the ASD [14] and Crystal Clear [15] methodologies, and was further explored in [12] to develop an adaptive project management framework. However, FRAME is novel in that it delineates an adaptive framework by defining a thorough metamodel which incorporates adaptability into the process framework. Moreover, FRAME models all the activities performed in an agile process, from the top-most enterprise management activities, to agile project management and the fine-grained agile practices and activities, all with the same simple self-similar metamodel. We have thus generalized the idea of adaptive frameworks to introduce an all-encompassing fractal metamodel for agile processes. FRAME provides a coherent whole, which not only covers existing methods, practices and activities, but also serves as a template for instantiation of new patterns

and method chunks in the context of agile development processes. To justify the application of FRAME as a process metamodel in instantiating agile processes, we have examined and categorized the relevant patterns of activity in seven prominent agile processes (six methodologies, and the ASDLC as a general lifecycle) based on the FRAME paradigm, and have thereby provided an initial repository of FRAME activity patterns. In fact, the FRAME repository illustrates the concretization of the abstract metamodel of FRAME in developing agile processes.

This paper is organized as follows: The proposed generic fractal process metamodel for agile methodologies (FRAME) is presented and described in Section 2; Section 3 shows the realization of FRAME in existing processes by providing a repository of activity patterns; and conclusions are presented in Section 4.

2 FRAME: FRactal Agile MEtamodel

FRAME models agile processes based on the idea of adaptive/incremental learning process frameworks, as introduced in [12], and [14]. To depict the FRAME scheme, four elements are considered for modeling agile processes, including:

- *Process Metamodel:* A template which models the overall stages in the development of a process.
- *Knowledge Flow:* The knowledge and experience grained and transferred during the development and instantiation of processes.
- *Basic Components:* The process elements that pass through the stages of the process metamodel in order to be refined and shaped into a process.
- *Process Environment:* The environment in which a process is developed, ranging from the enterprise environment to project and development environments.

The core element of FRAME is a process metamodel. It models agile processes with a basic self-similar component consisting of three main stages: Initiation, Learning Engine, and Termination. The Learning Engine itself consists of a loop of six stages: Initialization/Tuning, Freezing (consisting of Self-similar Execution and Finalization), Monitoring/Control, Buffering/Propagation, and Evaluation/Revision. The loop thus implements incremental learning at the process level. In summary, the Process Metamodel starts with the Initiation stage, during which initial knowledge is gained from the Process Environment. This ignites the Learning Engine in which the Process Metamodel learns to adjust itself with the Process Environment through executing a learning loop. Due to the incremental nature of learning, the Process Metamodel initializes itself with its current knowledge of the environment, enters the Freezing stage of Execution and Finalization, and Evaluates/Revises itself based on the knowledge gained through the Freezing stage and the changes in the Process Environment; this starts a new run of the Learning Stage. The intermediate Freezing stage, consisting of Self-similar Execution and Finalization stages, isolates the Process Metamodel from the volatility of the Process Environment. Since the rate of change in the Process Environment could be high, the duration of stay in this stage should not be long. This explains the parallel instances of Self-similar Execution. Iterative runs of the Learning

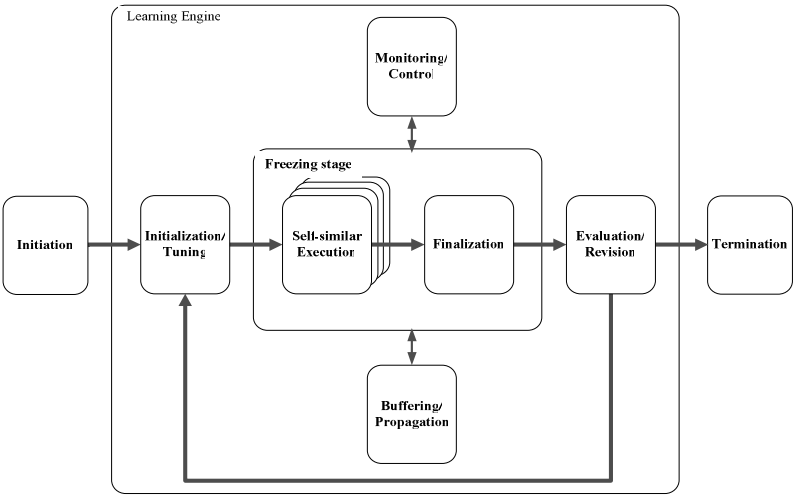

Fig. 1. Stages of FRAME

Engine evolve the Process Metamodel so that it fits the situation at hand. The stages of the FRAME process metamodel and the transitions among them are depicted in Figure 1. The fractal nature of FRAME derives from the Self-similar Execution stage, which accommodates nested instances of the FRAME process. This enables gradual refinement and separation of concern in the development of agile processes. The interaction between stages is established through the flow of knowledge between them. This means that in transition from one stage to the other, knowledge gained through one stage is transmitted to the next. The Knowledge Flow element is necessary for incorporating incremental learning into the model.

The Basic Components include the constituents on which the process metamodel operates. In fact, Basic Components are processed and modified throughout the stages of the process metamodel in order to shape agile processes. These constituents can be categorized in four groups (the four P's):

- *People*: consisting of individuals, teams, responsibilities, roles, skills, authorities, and communications.
- *Processes*: consisting of activities, tasks, methods, strategies, and related tools.
- *Products*: consisting of deliverables, artifacts, documents, and documentations.
- *Plans*: consisting of schedules, estimates, and stages.

The context in which agile processes are incorporated is the Process Environment. The process environment can be layered, with Sub-levels adding detail to super-levels. We have identified four basic levels, referred to as Process Levels:

- *Process Level 0 (Enterprise):* This level includes organizational infrastructures and processes. This is the place for embedding agile processes at organizational and enterprise levels, thus forming the "agile enterprise".

- *Process Level 1 (Project):* This level accommodates agility at the level of project-wide and project management activities, and abstracts umbrella activities performed in agile frameworks in the form of agile project management.
- *Process Level 2 (Development):* This level includes processes which incorporate agility into the development of products, forming an agile software process. Although the general term "agile software process" covers umbrella activities as well, FRAME considers umbrella activities at a higher level.
- *Process Level 3 (Construction):* This level embeds agility at the level of fine-grained activities, producing executable and tangible agile practices.

Thus, the development of agile processes in its generic form starts at enterprise and project infrastructures and continues through development and construction environments. However, agile process development could start at each of the Process Levels specified above, depending on the degree of agility targeted.

In summary, FRAME models agile processes via the four P's, refined and prepared through the stages of the Process Metamodel at each Environment level. In each stage transition, in addition to Basic Components, a Knowledge Flow is propagated through the Process Metamodel, aimed at transferring the current state of the Environment and the agile process developed so far. In a top-down approach, recursive application of FRAME at different process levels gradually shapes and fine-tunes agile processes in different environments. Hence, in the top-down approach, global and outline knowledge flows to lower levels, thus shaping the outlines of the processes. While in a bottom-up approach, local and specific knowledge flows to higher levels, thus adjusting the outlines specified. Moreover, at each level, the Learning Engine refines the processes at the current level. Therefore, applying the Learning Engine of FRAME at different levels shapes and adapts agile enterprise management, agile project management, and agile software development processes gradually, so that they fit their relevant environments.

The stages of the FRAME Process Metamodel will be examined through the rest of this section. It should be noted that FRAME does not specify a concrete process, but rather abstracts the activities taking place in agile processes. It therefore serves as a basic paradigm whose stages and core activities should be specialized with phases, stages and activity patterns in order to shape concrete processes.

2.1 Initiation Stage

The FRAME Process Metamodel starts with the Initiation stage at a specified process level. The main goal of this stage is to provide all the infrastructures and prerequisites for the activation of the Learning Engine. To this aim, high-level knowledge of the super- and current process levels is acquired, the current state of goals and requirements is identified, the initial and core configuration of the four-P's groups is specified such that they fit best to the current state, and the outline of the next stages is delineated. The core activities for realizing these steps, along with their outlines and the input/output Knowledge Flows, are shown in Figure 2. Due to the fractal nature of

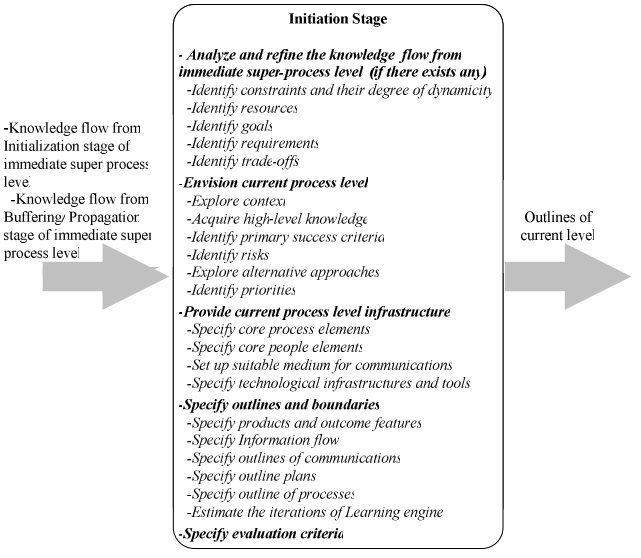

Fig. 2. Core activities of Initiation stage

FRAME, there are two knowledge flows: firstly, the knowledge flow from the super level, which carries information about the super-level and the outline boundaries which have been specified for the current level in the initialization stage of the super-level; and secondly, the changes made to this information as propagated by the Propagation/Buffering stage of the super-level.

2.2 Initialization/Tuning Stage

Two groups of activities are performed in this first stage of the Learning Engine: Initialization activities, which are performed in the first run of the Learning Engine at the current level; and Tuning activities, which are performed in subsequent runs (Figure 3). In short, in this stage, a subset of goals and requirements, which have been specified at the Initiation stage and/or modified at the Buffering/Propagation stage, are selected as targets. If this run is the first run, the outlines of the four P's, delineated during initialization, should be specialized in preparation for the execution of the Learning Engine. If this run is not the first, the settings should be tuned according to: the Knowledge Flow from the previous run; the changes that have occurred during the Freezing stage, propagated by the Buffering/Propagation stage of the current or the super process level; and experiences/modifications transferred from the immediate sub-level. The outline of the Execution stage should be specified, with special attention to parallelization.

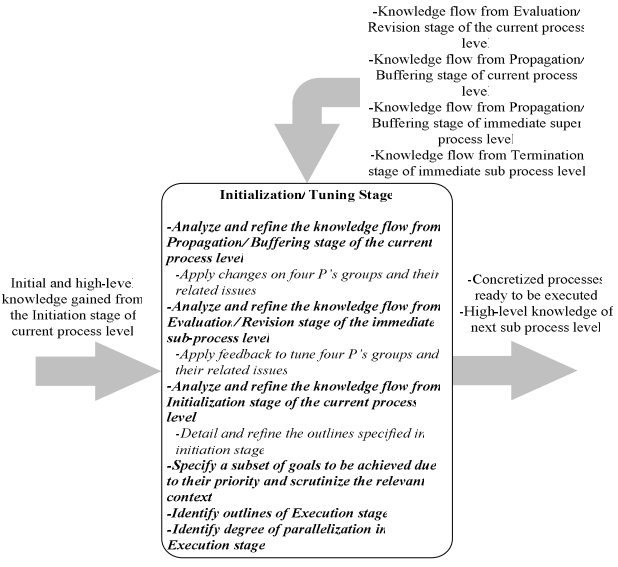

Fig. 3. Core activities of Initialization/Tuning stage

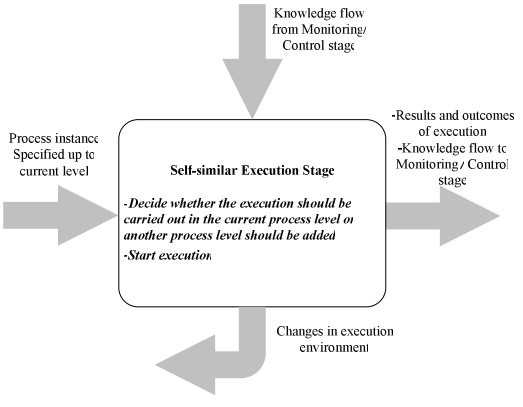

Fig. 4. Core activities of Self-similar Execution stage

2.3 Self-similar Execution Stage

Self-similar Execution is what gives FRAME its fractal nature (Figure 4). The transition of FRAME from one process level to another occurs at this stage through incorporating a nested (sub-level) instance of FRAME into the current level. Execution at different process levels results in different layers of agility; for instance, execution at the *Enterprise* level results in the instantiation of an enterprise-level process with its relevant four-P's groups, whereas execution at the *Project* level results in the instantiation

of the four-P's groups in a project framework aimed at running an agile project. If there is no need for a sub-level, execution begins based on the four-P's groups configured in previous stages.

At this stage, the process becomes isolated from its environment; meaning that no changes should occur in the settings that are passed to it. To control the settings, and to buffer changes occurring at the Freezing stages of the current and super levels, two other stages run in parallel with this stage: Monitoring/Control and Propagation/Buffering. Moreover, since the environment is dynamic, this stage should be kept short so that changes are considered as soon as possible. To control the duration of this stage, parallel instances of the stage are run simultaneously.

2.4 Finalization Stage

In this stage, the raw results and outcomes of Self-similar Execution are processed and refined (Figure 5). To this aim, the results of parallel instances of Self-similar Execution in the current run of the Learning Engine are verified, validated and integrated. Integration should also be applied to the results of current and previous runs of the Learning Engine at the same process level. The final outcome should be stabilized and fine-tuned to fit the current state of the environment.

2.5 Monitoring/Control Stage

The main goal of this stage (Figure 6) is to keep Self-similar Execution in line with the configuration set in the Initialization/Tuning stage. To this aim, configurations should be monitored constantly during execution. In case of deviation, appropriate mechanisms should be adopted to rectify the deviations. Problems impeding execution should

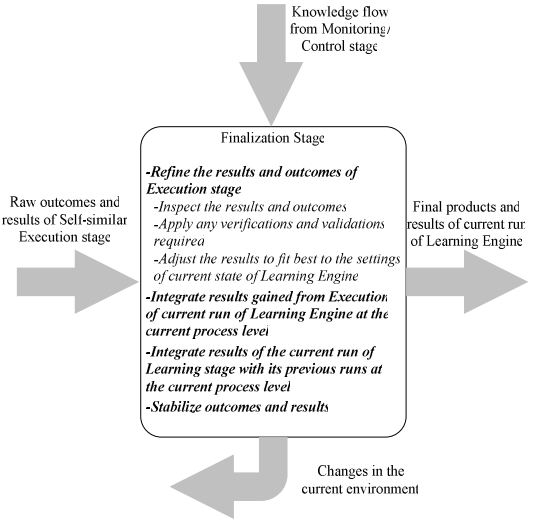

Fig. 5. Core activities of Finalization stage

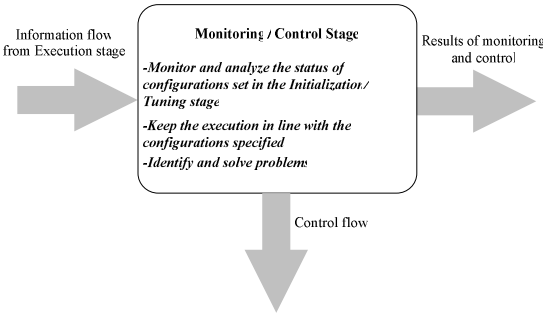

Fig. 6. Core activities of Monitoring/Control stage

be pinpointed and solved. Moreover, settings should be carefully scrutinized during execution, and their strengths and weaknesses should be identified to be considered in the Evaluation/Revision stage.

2.6 Buffering/Propagation Stage

In this stage (Figure 7), changes occurring during the Freezing stage are buffered, analyzed and propagated for consideration in the Initialization/Tuning stage. Moreover, this stage is an interface between different process levels: Changes occurring in the Freezing stage of the FRAME instance at a specific process level should be buffered to be considered in the next run of the Learning Engine of the current level. However, these changes are propagated instantly to the immediate sub-level. If the sub-level is in a state other than the Freezing stage, the changes are applied; otherwise, they should be buffered to be considered in the next run.

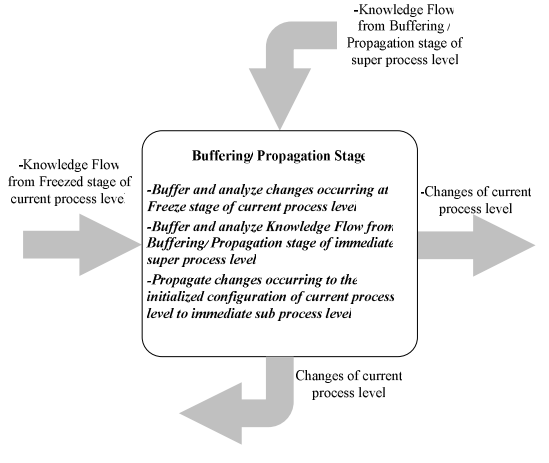

Fig. 7. Core activities of Buffering/Propagation stage

2.7 Evaluation/Revision Stage

The Evaluation/Revision stage aims at tailoring the process to fit the situation at hand (Figure 8). In this stage, the latest run of the Learning Engine is revised. To this aim, results of the Monitoring/Control stage are examined; the reliability of infrastructures and configurations is evaluated and compared with the results gained from the Learning Engine; problems impeding the full achievement of the goals are resolved; tuning points are specified to increase the consistency of the next run of the Learning Engine with the current state; decision is made on whether to start a new run of the Learning Engine, or to terminate the current run; and modifications made to the outlines specified at the current process level are propagated to the Initiation/Tuning stage of the super level.

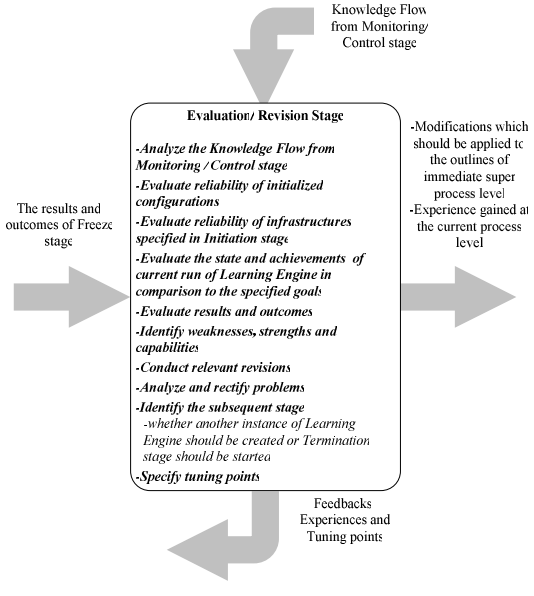

Fig. 8. Core activities of Evaluation/Revision stage

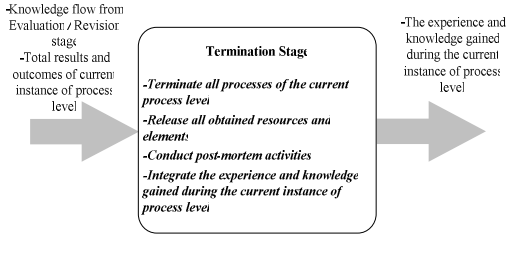

Fig. 9. Core activities of Termination stage

2.8 Termination Stage

In this stage, termination and wrap-up activities are performed (Figure 9); all resources and elements, including the four-P's groups, are released; post-mortem activities are conducted and lessons learned during the run of the whole process metamodel are compiled to be used in subsequent instances of super process levels.

3 Realization of FRAME in Agile Methodologies

As mentioned previously, FRAME exposes the paradigm behind agile processes, patterns and practices, and provides an abstract basis for engineering tailored-to-fit agile methodologies. In order to be used for developing agile processes, the abstract stages and outlines of activities should be specialized. Based on FRAME, instantiation of agile processes could be conducted in several ways. Based on the core activities outlined in FRAME, agile method chunks could be defined, which aid assembly-based composition of agile processes. Furthermore, the outlines of FRAME activities also serve as a substrate for definition of classes of agile processes and patterns for paradigm-based engineering purposes.

In this section, with the adoption of the paradigm-based SME approach, and using FRAME as a basic metamodel, we develop an initial repository of activity patterns. FRAME considers two main classes of agile activity patterns:

- *Stage class*: decomposing patterns based on the eight stages in the process metamodel. This class also provides sub-classes of patterns based on the outlines of the core activities specified.
- *Process-Level class*: providing a layering on patterns based on the environment to which they are applied. The layering provided by this class can be monotonically projected on lifecycle-scale activities.

Based on the above, we have analyzed seven prominent agile processes based on the FRAME paradigm. These processes include: 1) DSDM [13]; 2) SCRUM [5]; 3) XP [6]; 4) ASD [14]; 5) Crystal Clear [15]; 6) FDD [7]; and 7) Agile System Development Lifecycle (ASDLC) [8]. Activity patterns and tasks have been extracted from these processes as reusable patterns, expressed as FRAME stage- and process-level classes. Results have been tabulated in Table 1; numbers in parentheses refer to the processes from which the patterns have been extracted (as numbered in the above list). The repository provides a FRAME-based review of the seven processes: It shows their conformance to the FRAME meta-model, highlights their shortcomings and capabilities in covering FRAME classes, and provides opportunities for extending the processes with the patterns. The repository can be further detailed by decomposing the patterns into classes of core activities.

Table 1. FRAME repository of activity patterns

	Process Level 0 (Enterprise)	Process Level 1 (Project)	Process Level 2 (Development)	Process Level 3 (Construction)
Initiation	-Conduct Portfolio Management (7)	-Define Project Mission through JAD (4) -Develop Project Map (5) -Conduct Business Study through Facilitated Workshops (1) -Develop Release Plan/ Outline Plan/ Buffered Schedules (1,2,3,4,5) -Specify Time Boxes/ Time Frames (1,4) -Develop Spike Solutions/ Create System Metaphor/ Overall Architecture/ Initial Prototype (1,2,3) -Develop Product Backlog/ User Stories , Build Feature Lists (2,3,6) -Develop Rough Estimates/ Apply MoSCoW Rules (1) -Build the Core of Project Teams/ Build SCRUM Teams (2,3,4,5)	-Develop Overall Model/ Conduct Domain Walkthroughs (6) -Plan by Feature (6) -Develop Sprint Backlog/ Specify Mission Objectives (2,3,4) -Set up Environment for Development (7) -Conduct Card-based Planning/ Blitz Planning (2,5)	-Choose Work Package (6) -Design by Feature (6)
Initializing/Tuning	-Perform Exploratory 360 (5) -Develop Feasibility Prototype (1) -Shape and Fine Tune Methodology Conventions (5) -Establish Business Case for Projects -Apply Suitability Filters (1)	-Sprint Planning/ Adaptive Cycle Planning/ Recalibrate Release Plan/ Adaptive Planning (2,3,4,5) -Develop Initial Prototypes (1) -Workload (Re)Distribution/ Move around People (2,3,6) -Update and Refine Product Backlog/ User Stories/ Feature Lists (2,3,6) -Keep Sustainable Pace (3)	-Hold SCRUM Planning Meetings/ Iteration Planning/ Develop Fine-grained Plans (2,3,5) -Update/ Refine Sprint Backlog (2, 3) -Task Sign-up and Estimation/ Form Self-organized Teams (2,3) Assign Feature Sets to Chief Programmers (6)	-Assign Features to Class Owners (6) -Conduct Model Storming Sessions (7) -Form Team of Class Owners (6) -Test-driven Development (2,3)
Self-Similar Execution	-Concurrent Project Handling	-Development cycles/ Delivery Cycles/ Sprints (1, 2, 3, 4, 5, 7)	-Daily Cycles/ Concurrent Component Engineering (2,3,4,5)	-Integration Cycles/ Episodes (2,3,4,5)
Finalization	-Deliver & Operate Project Outcomes into Enterprise Environment	-Deliver nth Release (1, 2, 3, 4, 5, 7) -Conduct Productionizing Activities (3) -Conduct Comprehensive Validation Reviews(1)	-Conduct Regression Testing (1,2,3,4,5) -Deploy Pilot Release (1,2,3,4,5,7) - Conduct Pilot Test (7)	-Conduct Exploratory / Unit Testing (7) -Deploy Internal Release (2,3,5,7)
Monitoring/ Control	-Monitor by Executive Sponsor (5)	-Monitor by Domain Experts/ Apply SCRUM of SCRUMS (2,3)	-Monitor by Dedicated Care Taker/ Product Owner	-Hold 15-minute Daily Stand-up Meetings (2,3,5)
Propagation/ Buffering	-Share Mission Value among Project Community (4)	-Active Stakeholder Participation (1,2,3,4,5,7) -Specify Ambassador User (1,5) -Conduct Acceptance Testing/ Black-Box Testing (1,2,3,4,5,7)	-Conduct Scenario Testing / (7)	-Collective Code Ownership (3)
Evaluation/Revision	-Conduct Final Reflection (6)	-Reflect on the Delivery/ Reliability of Plans/ Methods/ Working Conventions, Hold Reflection Workshops (4, 5) -Conduct Confirmatory Testing (7) -Conduct Cycle Reviews /Holding Facilitated Customer Focus Group Sessions / Group Reviews, Hold Sprint Review Meeting (2,4) -Evaluate Project Velocity (3)	-Conduct Model/ Design Inspection, Refactor Model (6) -Conduct Iteration Completion Ritual / Reflect & Celebrate (5) -Product Demonstration to Users (1,2) -Conduct post – deployment reviews (4)	-Conduct Code Inspection/ Quality Reviews/ Reflect on Code Quality (4,5,6) -Refactor Mercilessly (3)
Termination	-Finalize Project Documentation/ Conduct Post-Project Activities	-Final Q/ A and Release (4) -Declare Project as Closed/ Conduct Post-Mortem Documentation and Review/ Compile Lesson Learned (3,4) -Conduct Brief Tour of System/ Wrap-up (3,5)	-Finalize Development Documentation (7) -Conduct Cycle Post-mortem (4)	-Promote to Build (6)

4 Conclusions

We propose FRAME as a generic metamodel based on which agile processes can be constructed. The novelty of FRAME is in that it strives to provide a comprehensive model for agile methodologies, covering both development activities and umbrella activities, through providing a fractal process metamodel.

Based on FRAME, we have developed a FRAME repository of relevant activity patterns extracted from agile processes. This provides a substrate for paradigm-based engineering of agile methodologies. The outlines and core activities specified in FRAME can also be used in the definition and extraction of method-chunks, and ultimately utilized for assembly-based engineering of agile processes.

The generic process metamodel of FRAME is not fully concretized in current agile methodologies. This leaves the instantiation of the abstract core activities and metamodel – in the form of patterns, method chunks, and methodologies empowered by FRAME's potentials – as an open issue. This may also facilitate the improvement of existing agile methodologies. Extended by activity patterns, FRAME provides a rich repository for the use of CAME tools [10] in the context of engineering agile processes. The stages and core activities of FRAME, if further detailed, can also provide a comprehensive benchmark for criteria-based analysis, comparison, and evaluation of agile methodologies.

References

1. Ramsin, R., Paige, R.F.: Process-Centered Review of Object Oriented Software Development Methodologies. ACM Computing Surveys 40(1), 1–89 (2008)
2. Beck, K., et al.: Manifesto for Agile Software Development, http://agilemanifesto.org/
3. Alliance, A.: Agile Principles, http://agilealliance.org/
4. Ralyté, J., Brinkkemper, S., Henderson-Sellers, B.: Situational Method Engineering: Fundamentals and Experiences. Springer, Heidelberg (2007)
5. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice-Hall, Englewood Cliffs (2001)
6. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley, USA (2004)
7. Palmer, S.R., Felsing, J.M.: A Practical Guide to Feature-Driven Development. Prentice-Hall, Englewood Cliffs (2002)
8. Ambler, S.W.: The agile system development lifecycle (2006), http://www.ambysoft.com/essays/agileLifecycle.html
9. Tasharofi, S., Ramsin, R.: Process Patterns for Agile Methodologies. In: Situational Method Engineering, Fundamentals and Experiences, pp. 222–237. Springer, Heidelberg (2007)
10. Niknafs, A., Ramsin, R.: Computer-Aided Method Engineering: An analysis of existing environments. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 525–540. Springer, Heidelberg (2008)
11. OMG. Software and Systems Process Engineering Metamodel Specification (v2.0). Object Management Group, OMG (2007)

12. Hasani Sadi, M., Ramsin, R.: APM^3: A project management methodology metamodel for agile methodologies. In: Proceedings of the 8th International Conference on Software Methodologies, Tools and Techniques, SoMeT 2009 (to be published 2009)
13. DSDM Consortium. DSDM Business Focused Development, 2nd edn. Addison-Wesley, Reading (2003)
14. Highsmith, J.: Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House, Newyork (2000)
15. Cockburn, A.: Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley, Reading (2004)