A Framework for Extending Agile Methodologies with Aspect-Oriented Features

Maryam Gerami, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology, Tehran, Iran
m_gerami@ce.sharif.edu, ramsin@sharif.edu

Abstract—**Aspect-Oriented Software Development (AOSD) concentrates on determination and modularization of orthogonal concerns that cut across system modules. An aspect-oriented extension framework helps transform existing methodologies into processes which support AOSD. The authors propose a framework for extending agile methodologies with AOSD features. The stages required for constructing the framework have been identified by scrutinizing existing aspect-oriented methodologies and practices in order to extract the AOSD-specific stages required in each phase of the agile development process. Analysis shows that a methodology extended through this framework can be superior in agility in comparison to a general aspect-oriented methodology.**

Keywords-Aspect-Oriented Software Development; Extension Framework; Agile Methodology

I. INTRODUCTION

Aspect-Oriented Software Development (AOSD) is regarded as a promising method for separating and modularizing cross-cutting concerns at a higher level than that usually provided in object-oriented development–where implementation of such concerns can result in *scattering* and *tangling* problems [1]. Detection and separation of cross-cutting concerns during early phases of software development improves software evolvability and modifiability. However, due to the relative novelty of the Aspect-Oriented (AO) approach, researchers have concentrated more on AO programming and modeling rather than on AO methodologies and processes. Consequently, organizations still suffer from a lack of concrete methodology support for AOSD. On the other hand, the ever-increasing need for rapid software development and higher product quality has led to the popularity and widespread adoption of *agile* methodologies [2]. Agile methodologies embrace change and pay special attention to satisfying customer change requests, and can therefore greatly benefit from the AOSD approach. However, little work has been done on the integration of these two approaches. The authors believe that an integrated approach, leading to the notion of aspect-oriented agile processes, can significantly increase AOSD's potential for adoption in the industry.

Our research, as reported herein, aims at developing a process for adapting agile methodologies to aspect-oriented development contexts. An extension framework is proposed for this purpose, shaped and refined through scrutinizing existing AO methodologies and practices, and extracting/abstracting the necessary AO activities and products for each generic phase of the agile development process. This extension framework can be used to create new AO agile methodologies, as well as to add AO support to existing agile processes. Adding AO support can eliminate the need for replacing the methodology that is currently being practiced in an organization: such naturalized methodologies are usually better suited to the requirements and constraints of the organization and its typical projects, and replacing them with a new AOSD methodology can have extensive adverse repercussions, not to mention the cost and time overheads involved. The proposed framework enables developers to extend an existing development process with AO features instead of replacing it.

A considerable body of research and practice focuses on the different features of AOSD methods [3, 4]. However, research on adding AO features to non-AO methodologies has been limited: In [5], a method is proposed for unifying agile and AO requirements analysis approaches; in [6], a combination of techniques is used for enhancing support for AOSD, including the specification of the method components required. As an example of using extension frameworks for enhancing agile methodologies, the work reported in [7] proposes an extension framework for adding MDA features to existing agile methodologies through a Method Engineering (ME) approach.

The framework that we propose utilizes the *extension-based* approach of Situational Method Engineering (SME) [8], which aims to construct methodologies to fit specific project situations according to predetermined organizational settings. The proposed framework covers the entire software development lifecycle; it thus determines which basic features an aspect-oriented agile development process should possess, and can therefore also act as a benchmark for the evaluation and comparison of different methodologies. The Agile System Development Lifecycle (ASDLC) [9] has been used as the basis for defining the activities of the proposed extension framework; ASDLC is an abstract agile development lifecycle which covers the processes of different agile methodologies. Thus, the extension framework becomes independent of any concrete agile methodology. In order to apply the extension, an agile methodology should first be mapped to ASDLC: Applying the extension framework to the selected methodology is performed based on the correspondence between the phases of the selected methodology and those of the ASDLC. In order

to scrutinize and evaluate the proposed extension framework, an analysis has been conducted to make sure that the extension process does not compromise agility; a more general criteria-based evaluation has also been performed in order to highlight the strengths and weaknesses of the framework.

The rest of this paper is organized as follows: Section II presents the proposed extension framework; Section III points out the preconditions that should be met for the extension framework to be applicable in a specific methodology-extension context; Section IV analyzes the effects of applying the framework on the agility of methodologies; Section V provides a criteria-based analysis of the extension framework; and Section VI provides the conclusions and outlines several directions for furthering this research.

II. PPROPOSED EXTENSION FRAMEWORK AND PROCESS

A detailed description of the proposed extension framework will be provided in this section. The framework is the result of investigating and analyzing different resources and observations on both AO and agile development approaches. The extension framework was constructed based on the results of the conducted analyses, aiming to provide a template for enriching a base methodology with AO features. To work out an effective and optimized framework, certain refinement criteria were taken into account during its construction, such as "minimal change to the base methodology". The framework was then revised and refined iteratively through application to selected agile processes and meta-models.

In order to provide developers with a prescribed procedure for extending agile methodologies, the proposed extension framework is applied as part of a proposed *methodology extension* (method engineering) and *methodology execution* process. The methodology extension/execution process, as depicted in Fig. 1, is composed of four phases; the extension framework is applied in the Fusion sub-phase of the third phase, and specifies the AO activities and products typically present in aspect-oriented processes. The extension framework is thus applied through the extension process to fuse aspect-oriented features into the base methodology. The extension and execution phases, and the framework applied, will be described in more detail throughout the rest of this section.

A. Project Initiation

Detailed feasibility study and justification are performed in this phase. The extension and execution effort is considered a project in its own right. The project's financial needs are estimated and resources are allocated to the project. Time and cost limitations are identified and set as project constraints. Since agile methodologies prescribe close communication with the users, one or more user representatives are identified to participate in the development team.

Team organization and assignment of project roles to team members is also performed in this phase. In particular, methodology engineers, AOSD experts, and domain experts are the roles necessary in the methodology extension context. This phase also deals with the preparation of the development and user environments: the hardware/software infrastructure and tools necessary for the initiation of the project are identified and installed.

B. Development Process Selection

In this phase, methodology engineers identify the base development methodology suitable for the project at hand, mainly based on the project scope and size. Typically, the base methodology selected for extension is the one currently being used in the organization.

C. Adding Support for Aspect-Oriented Development

This pivotal phase is where the methodology identified in the previous phase is extended with AOSD support. This phase consists of four sub-phases, as described below.

1) Modularization Unit and Modeling Language Selection

Concerns are modularized based on use-cases, scenarios, or other similar units. A modularization unit does not change during the development process and must be identified and finalized in this sub-phase. The modeling language is selected based on the AO-specific features that should be supported, the modularization unit, and the development approach.

2) AO Notation Support

AO modeling approaches are less mature, and they are not fully supported by most modeling languages; AOSD thus faces serious challenges at the modeling level. In this sub-phase, AO features are added to the modeling language notation, and are then used as a standard throughout the project.

3) Tool Selection

The use of appropriate, cost-effective tools has a positive effect on the velocity and simplicity of development; hence, it often reduces development costs. The selected language and the development process are the main factors that influence tool selection.

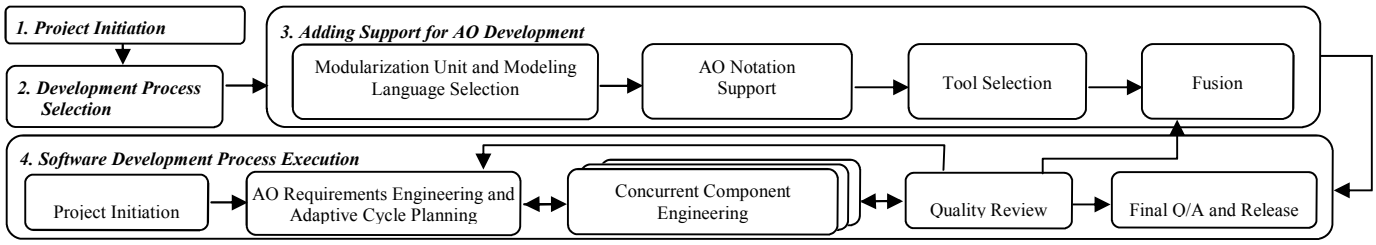

Figure 1. Proposed methodology extension/execution process–an extended version of the ASD methodology has been shown in the last phase as an example

4) Fusion

The last sub-phase of methodology extension is transforming the selected agile methodology into an aspect-oriented variant. In order to add AOSD support to the development phases, additional or transformed tasks are specified for each generic development phase. The proposed extension framework, depicted in Fig. 2, would affect three generic phases of agile methodologies, including requirements engineering, architecture design, and detailed design:

- Requirements Engineering: AO requirements engineering deals with identifying and modeling the concerns (cross-cutting and non-cross-cutting) and their relationships. Specification and modeling of the relationships among concerns helps identify the candidate aspects. Certain priority criteria are also defined to be used for handling the conflicts that may arise during the composition of aspects and other concerns.

- Architecture Design: AO architectural modeling of the system is the main concern of this phase, which may lead to the identification of new aspects. The composition of architectural aspects to realize system functionality, often results in conflicts among concerns that should be handled by applying the priority criteria obtained in the requirements engineering phase.

- Detailed Design: Detailed aspect-oriented structural and behavioral modeling of the system is performed in this phase. Design-level aspects and other concerns are composed and possible conflicts are handled to create a detailed model of the system that realizes the required functionality. It is preferable to validate the design against stakeholders' concerns to ensure that all the requirements are thoroughly met.

- Umbrella activities: In parallel with development activities, umbrella activities are performed in order to monitor and manage the project. They include quality management, project management, tools and resource management, training, risk management, software configuration management, and infrastructure management.

The extension framework of Fig. 2 has been defined in terms of its impact on the development phases of the base methodology. The three phases shown in this figure correspond to the extension points of the base methodology, and the activities specified in each are the AO activities that should be added/enhanced in order to instill adequate AOSD support into the base methodology. The base methodology is thus *fused* into the proposed framework.

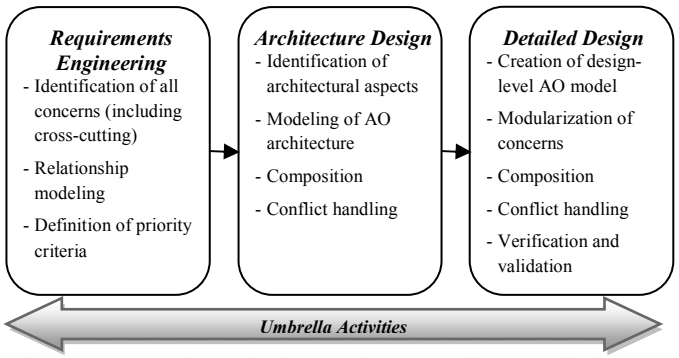

Figure 2. Proposed extension framework, expressed in terms of its impact on generic software development phases

In order to better specify the extension points in an agile methodology, the proposed extension framework has been applied to the Agile System Development Lifecycle (ASDLC), which defines a general process framework for agile methodologies [9]. AOSD activities and products are thus added to an abstract agile lifecycle, and the resulting AO-enriched generic process is a concrete manifestation of the extension framework proposed, and can therefore act as a methodology extension framework in its own right: Existing agile methodologies can be fused into this process by determining their mapping to the ASDLC, and then extending them – at their corresponding extension points – with the AOSD features added to the ASDLC. ASDLC was chosen because it is the most widely known generic agile development lifecycle in use. As depicted in Fig. 3, four phases of the ASDLC need to be extended with AO stages:

a) Iteration -1

The definition of the business opportunity, development of a viable plan for the project, and assessment of feasibility are the main activities of iteration -1. Conventional artifacts of this phase, such as the requirements specification document, are enhanced to cover aspectual requirements as well. This phase encompasses the following AO-extended stages:

1. Application of suitability filter: The suitability filter is employed to make sure that using aspect-oriented techniques does indeed bring about a positive outcome and a desirable level of return-on-investment. Cost-benefit study of the project at hand is performed with respect to important factors such as constraints, project requirements, and the required level of AOSD skills and experience.

2. Working conventions refinement: In this stage, methodology constraints, standards, and rules are refined based on the specific needs of AOSD.

3. Requirement and aspect identification: In this stage, an AOSD expert identifies the main aspects of the system, and creates a high-level architecture for the system. Identified requirements and aspects are assigned to construction iterations, and requirements which can be implemented in parallel are identified and assigned to separate teams. Thus, overall cycle planning is performed in this stage.

b) Iteration 0

Identifying the system scope, forming the team(s), creating an initial architecture, setting up the development environment, and estimating the project's cost and duration are the main activities of this phase. The plan, aspect model, and requirements specification are also produced in this phase. This phase encompasses the following AO-extended stages:

1. Concern specification and identification: The modeling language and tools selected in the previous phase are used for concern identification and modeling.

2. Aspect identification: Concerns that appear to have scattering and/or tangling properties in the requirements documentation are recognized as aspects. Various methods are proposed for detection of candidate aspects, such as the methods introduced in Theme [10] and AOSD/UC [11].

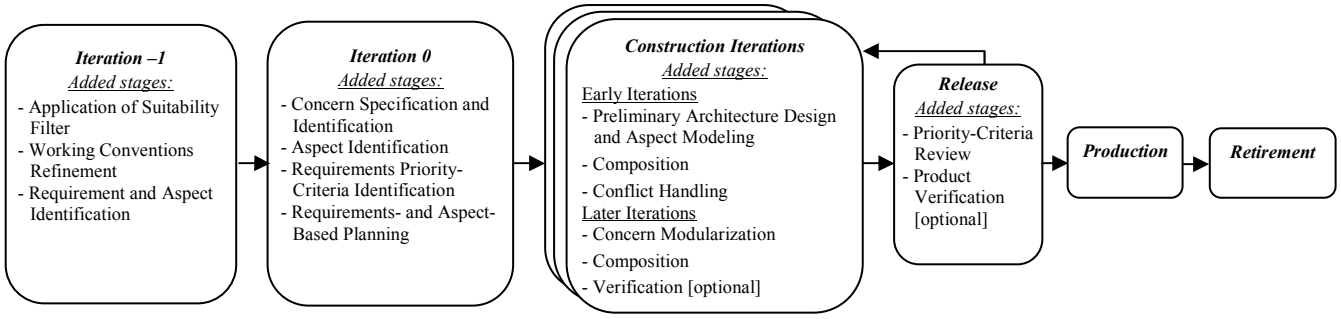

Figure 3. Results of fusing ASDLC into the proposed extension framework

3. Requirements priority criteria identification: in this stage, priority criteria are determined and the requirements assigned to each iteration are prioritized accordingly. Prioritized requirements are used as a basis for handling the conflicts that may arise when composing the aspects and base requirements.

4. Requirements- and aspect-based planning: In this stage, the duration and number of iterations are determined with respect to the requirements and aspects detected.

c) Construction Iterations

Analysis, design and implementation of the software functionality, quality assurance, and delivery of the working software, are carried out in these iterations. We have divided the iterations of this phase into earlier and later iterations. The architecture of the software is created during earlier iterations, while detailed design-level modeling is performed in later ones.

- Earlier iterations

1. Preliminary architecture design and aspect modeling: as mentioned before, this stage is devoted to the creation of a preliminary architecture for the subsystems. During construction iterations, new aspects are also detected and added to those previously identified. The modeling of each aspect is independent from other concerns, and the modeling language used is required to support the representation of aspects.

2. Composition: After architectural aspects are detected and modeled, they should be integrated into the main models, so as to realize the functionality of the system at the architectural level, resulting in an integrated architecture model.

3. Conflict handling: The tradeoffs existing among different requirements of the system may lead to conflicts during the composition of aspects. In this stage, high-level decisions are made for handling the conflicts. These decisions may result in the merging or removal of some aspects.

- Later iterations

1. Concern modularization: For creating an appropriate design, the design repository is first searched in order to make utmost reuse of existing design components. However, if reuse is not possible, the design of system components is carried out from scratch. Design of concern modules and identification of finer-grained concerns are also performed in this stage.

2. Composition: In this stage, the composition of aspects and other concerns is performed. Each of the composition specification documents produced in this stage should represent a detailed model of composition. Conflicts that occur during composition are handled based on the priority criteria.

3. Verification [optional]: Once the previous stages are completed, formal concern analysis techniques are used for verification of the design. Moreover, the design must be validated against the requirements.

d) Release

The latest construction iterations are reviewed and, if necessary, changes are made to the process. The reliability of the hardware/software infrastructures and configurations are examined, construction problems are analyzed, and the settings required for the next iteration are applied. In addition to the usual stages, the AOSD approach imposes certain extra stages:

1. Priority-criteria review: Priority criteria are updated based on the change requests received during the project.

2. Product verification [optional]: The implementation is verified against the design. Since serious modularity and maintainability problems may creep into the product due to non-conformance of the implemented software to the design, performing this stage may become essential. However, examination of the compliance of the implementation with design decisions can be performed in tandem with implementation activities during construction iterations; assigning a separate stage is therefore optional.

D. Software Development Process Execution

After fusing the selected methodology into the framework, the methodology can be enacted. As an example, Fig. 1 depicts a scenario in which the ASD methodology [12] has been integrated with AOSD features and then executed.

III. APPLICATION PRECONDITIONS

In this section, we define the preconditions that the base methodology should satisfy in order for the extension framework to be applicable. The only agile principle that may potentially be in conflict with the extension process, is giving excessive weight to face-to-face conversation in lieu of models. Following this principle can result in model-phobic methodologies; whereas models, especially those at the design level, are a major feature of AOSD. The extension framework proposed can be applied to agile methodologies that allow the rational use of models. In order to determine how agile methodologies differ in aspect-oriented extensibility, we reviewed eight agile methodologies: ASD, FDD, DSDM, AM, dX, XP, Scrum, and Crystal Clear [2]. Based on our observations, it seems that our extension framework can be used for extending these methodologies.

IV. Analysis of Agility

In order to validate the extension framework, it should first be shown – through application of suitable agility criteria – that it does not adversely affect agility. For evaluating the agility of the extended methodology resulting from the application of the proposed framework, a comprehensive set of criteria is needed. A recent development in this context is the advent of the 4-DAT evaluation framework; 4-DAT tries to address all aspects of agility by defining four dimensions: Method scope, agility features, characterization of agile values, and software process characterization [13]. We have applied the 4-DAT framework in order to evaluate the impact of the extensions made by the proposed framework on the agility of base methodologies; the results, as tabulated in Table I, show that applying our proposed framework does not have an adverse effect on agility, and can even promote agility in certain aspects.

V. Criteria-Based Aspect-Oriented Analysis of the Proposed Extension Framework

In order to expose the strengths and weaknesses of the proposed extension framework, it should be analyzed according to suitable evaluation criteria. These criteria were obtained through investigating several aspect-oriented approaches. The preliminary criterion set was applied to the proposed extension framework, and then iteratively refined based on the evaluation results. In order to quantify the analysis results, the *Feature Analysis* [14] approach has been used. Based on this approach, a number of quantification categories are defined, and each criterion is assigned to one of these categories based on its disposition. For our criterion set, four types of categories have been introduced: A *Simple Form* (P), an *Enumerated Form* (E), a three-level *Scale Form* (S), and a *Descriptive form* (D). In Table II, each evaluation criterion has been introduced along with its category and a short definition. The results of applying the criteria to the proposed extension framework are shown in the last column of the table. The results show that the proposed extension framework does indeed provide adequate coverage of AOSD features, while preserving the essential features of the base methodology.

VI. Conclusions and Future Work

The aspect-oriented extension framework proposed herein aims to provide the means for engineering aspect-oriented agile software development processes through extending existing methodologies. We have shown how existing agile methods that satisfy the predefined prerequisites are fused into the framework to obtain an aspect-oriented agile methodology. Moreover, analysis shows that augmenting agile methodologies with aspect-oriented features using the proposed extension framework has the advantage of increasing the agility of the methodology in certain aspects.

Future research can be focused on refining the framework: A more detailed specification of the activities and roles involved in each stage is particularly required. In addition, applying the framework to agile methodologies in a practical SME context seems to be a worthwhile endeavor, since it helps analyze the framework as to its capabilities and limitations.

Acknowledgments

We wish to thank Iran Telecommunications Research Center (ITRC) for sponsoring this research, and Mr. Massood Khaari for assisting in the refinement of this paper.

TABLE I. Impact of the Proposed Extension Framework on Dimensions of 4-DAT

	Criterion	*Extension Effect*
Scope	***Project size***	The higher level of separation of concern leads to a higher level of reusability and better parallelization of activities among development teams. Therefore, the organization can undertake larger projects.
	Team size	Participation of one or more AOSD experts in teams.
	Development Style	No effect - the proposed framework adds a number of stages to the phases, but it does not change the phase transitions.
	Code style	Organizing the code in modules encapsulating only one concern
	Technology environment	Using aspect-oriented compilers and tools that support AO activities
	Physical environment	No effect
	Business culture	No effect
	Abstraction mechanism	Object-oriented - advices are woven into object-oriented classes
Agility Features	***Flexibility***	Priority criteria review and refinement, facilitation of code modification, and reduction of change propagation (enhancing modularization as compared to OOD)
	Speed	Higher level of modularization results in better code reusability, and the limited scope of change propagation facilitates the addition of increments. Also, higher level of parallelization of activities increases the development speed.
	Leanness	No effect - time limits are not affected by the framework and development tools are used only if they are cost-effective.
	Learning	Quality review results are used in process improvement
	Responsiveness	Responds to change in priority criteria
Agility Values	***Individuals and interactions over processes and tools***	No effect
	Working software over comprehensive documentation	No effect
	Customer collaboration over contract negotiation	No effect
	Responding to changes over following a plan	No effect
	Keeping the process agile	No effect
	Keeping the process cost effective	No effect
Software Process	***Development process***	Extension of the development process based on the proposed framework, and determination of rules and standards for definition of advices and join-points
	Project management process	Higher separation of concern affects responsibility assignment to individuals and teams. Aspects should be taken into account in project planning.
	Software configuration control process/support process	Aspect-related artifacts should be taken into account in configuration management.
	Process-management process	No effect

TABLE II. EVALUATION CRITERIA, AND THE EVALUATION RESULTS FOR THE PROPOSED EXTENSION FRAMEWORK

	Criterion	Category	Definition	Value
General	***Preservation of Philosophy***	D	No adverse effect on the philosophy of the base methodology, including the development style, and the number, nature, and sequence of phases and tasks	No new products are introduced. Style is not affected. Number of phases is not changed, unless the base methodology does not cover phases that should be extended by the framework.
	Preservation of Application Domain	S	No adverse impact on base methodology's application scope (1) Limiting application domain; (2) Preserving application domain (3) Extending application domain	(2)- AO is mainly based on OO, and extending a class of OO methodologies – such as agile methodologies – by using the proposed framework would not limit the application domain.
	Preservation of Traceability	P	No adverse effect on the traceability of artifacts to their source of origin, and traceability between software artifacts	Supported- homogeneous treatment of concerns helps preserve traceability (if supported by the base methodology).
	Support for Verification and Validation	E	Support for verification and validation of requirements against stakeholder concerns, and verification of architecture, design and intermediate artifacts against requirements	Support is provided for verification and validation of architecture and design artifacts against requirements; other verification activities supported by the base methodology are not undermined.
	Support for Evolvability	D	Simplicity of evolving (adding/removing/changing) software development artifacts while keeping their structural and behavioral purpose	Better modularization of requirements, analysis, design and consequently implementation artifacts (compared to OO)
	Support for Scalability	D	Applicability of the method to large and small projects	Enhanced due to higher level of modularization
	Coverage of Generic Software Development Lifecycle	E	Phases of the generic software development lifecycle that are covered by the framework	Project Initiation, Requirements Engineering, Architecture Design, and Detailed Design
	Support for Tool Selection	P	Support for investigation and selection of appropriate cost-effective tools	Supported
	Support for Umbrella Activities	S	(1) No support for umbrella activities (2) Assigning the definition of activities to software/method engineer (3) Full support for umbrella activities	(2)
Requirements Eng.	***Functional and Non-functional Cross-cutting Concern Identification***	S	(1) No support for concern identification (2) Support for non-functional cross-cutting concerns (3) Support for functional and non-functional cross-cutting concerns	(3)
	Support for Mapping	P	Support for mapping aspectual requirements to architecture or design elements or design decisions	Not supported
	Homogeneity in Treatment	P	Homogeneous treatment of cross-cutting and non-cross-cutting concerns	Supported
	Support for Conflict Handling	P	Support for analyzing tradeoffs and handling conflicts among concerns	Supported
	Composability	P	Support for requirements-level composition of concerns using composition semantics and join-point modeling	Not supported, as it could reduce agility.
Architecture Design	***Composition Modeling***	P	Detailed modeling of composition of aspects and other components	Supported
	Concern Independence	P	Independence of concern specifications, including concern composition constraints (helps enhance concern reusability)	Supported
	Composition	P	Support for architecture-level composition of aspects and other architectural elements	Supported
	Composition Constraints Identification	P	Support for identification of assumptions and constraints of concern composition	Not supported
	Architectural Tradeoff Analysis	P	Analysis of tradeoffs arising among architecture-level concerns	Supported
	Architecture-Level Validation	P	Testing of architecture to ensure that stakeholder concerns are realized	Supported
Detailed Design	***Concern Separation Level***	S	(1) Undefined; (2) Asymmetric; (3) Symmetric	(3)
	Prevention from change propagation	D	Simplicity of changing parts of models while not affecting other model elements	High level of separation of concern and modularization helps in reduction of change propagation.
	Reusability	D	Support for reusing produced components in other systems	Enhanced due to resolution of scattering/tangling problems
	Design-Level Concern Modeling	P	Detailed structural and behavioral modeling of all types of concerns	Supported
	Design-Level Composition Modeling	P	Detailed modeling of design-level composition, including specification of join-point and point-cut expressions and advices	Supported
	Design-Level Validation	P	Validation of the design models against stakeholder concerns	Supported

REFERENCES

[1] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kuleska, C. Lucena, and A. Von Staa, "Modularizing design patterns with aspects: a quantative study," Proc. 4th International Conference on Aspect-Oriented Software Development (AOSD), 2005, pp. 3-14.

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

[3] R. Chitchyan, et al., "Survey of analysis and design approaches," AOSD-Europe, Survey Report AOSD-Europe-ULANC-9, 2005.

[4] S. Op. de Beeck, et al., "A study of aspect-oriented design approaches," Technical Report CW435, Katholieke Universiteit Leuven, 2006.

[5] J. Araujo and J.C. Ribeiro, "A scenario and aspect-oriented requirements agile approach," International Journal of Computing Science and Applications, Vol. 5, No. 3b, 2008, pp. 69-92.

[6] B. Henderson-Sellers, R. France, G. Georg, and R. Reddy, "A method engineering approach to developing aspect-oriented modeling processes based on the OPEN process framework," Information and Software Technology, Vol. 49, No. 7, 2007, pp. 761-773, doi: 10.1016/j.infsof.2006.08.003.

[7] F. Chitforoush, M. Yazdandoost, and R. Ramsin, "Methodology support for the model-driven architecture," Proc. 14th Asia-Pacific Software Engineering Conference (APSEC), 2007, pp. 454-461, doi: 10.1109/ASPEC.2007.58.

[8] Ralyté, J., S. Brinkkamper, and B. Henderson-Sellers, Eds., Situational Method Engineering: Fundamentals and Experiences. Springer, 2007.

[9] S.W. Ambler, "The Agile System Development Life Cycle (ASDLC)," available online: http://www.ambysoft.com/essays/agileLifecycle.html [Accessed: May 2010].

[10] S. Clarke, and E. Baniassad, Aspect-Oriented Analysis and Design: The Theme Approach, object Technology Series, Addison-Wesley, 2005.

[11] I. Jacobson, and P.-W Ng, Aspect-oriented Software Development with Use Cases, Addison Wesley Professional, 2005.

[12] J. Highsmith, Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House Publishing Co., 2000.

[13] A. Qumer and B. Hendersson-Sellers, "An evaluation of the degree of agility in six agile methods and its applicability for method engineering," Information and Software Technology(IST), Vol. 50, No. 4, 2008, pp. 280-295, doi: 10.1016/j.infsof.2007.02.002.

[14] B. Kitchenham, S. Linkman, and D. Law, "DESMET: A methodology for evaluating software engineering methods and tools," Journal of Computing and Contrological Engineering, No. 8, 1997, pp. 120-126, doi: 10.1049/ccej:19970304.