2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems

Process Patterns for Aspect-Oriented Software Development

Massood Khaari, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
khaari@ce.sharif.edu, ramsin@sharif.edu

***Abstract*—Focusing on aspects during early stages of the software development lifecycle has received special attention by many researchers, leading to the advent of numerous Aspect-Oriented Software Development (AOSD) methods. This has consequently given a relatively high level of maturity to aspect-oriented processes. Process patterns, on the other hand, have been adopted as suitable mechanisms for defining or tailoring processes to fit specific organizational/project requirements. Process patterns, which essentially are reusable process components extracted from successful processes and practices, can be used to engineer new software development methodologies or to enhance existing ones.**

We propose a generic Aspect-Oriented Software Process (AOSP), constructed through studying and abstracting prominent aspect-oriented processes. Based on the proposed AOSP, process patterns are provided which incorporate well-established aspect-oriented practices for different development stages. By employing specific process evaluation criteria, the characteristics of these patterns have been analyzed.

Keywords-component; Process Patterns; Situational Method Engineering; Aspect-Oriented Software Development

I. INTRODUCTION

Process patterns are attracting growing attention for engineering software development processes [1, 2]. They are created by searching for processes that have been proved to be successful in practice, and abstracting away their low-level details in order to make them reusable in other situations and contexts. They therefore provide a suitable mechanism for composing processes based on the specific requirements enforced by organizations and projects. This particularly goes well with the *assembly-based* approach of Situational Method Engineering (SME) which proposes the idea of reusing existing method parts to construct new methodologies or enhance existing ones [3]. This approach takes advantage of a repository of reusable method fragments from which method engineers can select and assemble appropriate elements to create a custom methodology. Process patterns are thus suitable for use as method fragments in this context.

Aspect-Oriented Software Development (AOSD), on the other hand, has long been hailed as an effective means for addressing crosscutting concerns in software development. AOSD processes have matured over the years and have attracted widespread attention in the research community; however, the experience and knowledge gained has not been compiled, abstracted and distilled in the form of process patterns.

We propose the Aspect-Oriented Software Process (AOSP) as a generic process model for aspect-oriented software development. The primary aim of AOSP is to provide a generic pattern-based framework to support method engineering in the context of AOSD. It can also be used as a means for assessing aspect-oriented (AO) processes. Based upon AOSP, a set of process patterns is also proposed; these patterns have been extracted through studying prominent AO methodologies (as well as AO practices offered for different phases of software development), abstracting them, inspecting their commonalities, and extracting well-established AO process components. It is worth noting that AOSP is not itself an AO methodology. Rather, using its constituent patterns as process components, the AOSP can be used by method engineers as a general framework for engineering bespoke aspect-oriented methodologies. We also introduce coherent sets of analysis criteria for assessing the validity of the patterns proposed.

The rest of the paper is organized as follows: Section 2 provides an overview of the research background on process patterns and AOSD; Sections 3 and 4 present the proposed AOSP and describe the constituent process patterns; Section 5 introduces a set of analysis criteria, based on which an analysis on the AOSP and the process patterns is conducted; and Section 6 provides a summary, and briefly touches on potential strands for furthering this research.

II. RELATED RESEARCH

This section provides a brief account of the research conducted on process patterns and AOSD.

A. Process Patterns

Application of a comprehensive set of process patterns was first conducted by Ambler in the domain of object-oriented (OO) software development; Ambler used the patterns to form a process framework, called Object-Oriented Software Process (OOSP) [1]. His proposed OO process patterns are classified, based on their level of abstraction and granularity, into three levels of *phase*, *stage*, and *task*. A *task* process pattern defines a fine-grained activity to perform a small part of a process. *Stage* patterns define the activities required to accomplish a single stage of a process (usually in an iterative fashion), and are typically composed of a number of task (or nested stage) process

978-0-7695-4005-4/10 $26.00 © 2010 IEEE
DOI 10.1109/ECBS.2010.33

patterns. *Phase* patterns consist of two or more stage process patterns, and collectively form the high-level software development lifecycle.

Catalysis [2] is one of the first methodologies to use process patterns as frameworks for adapting to different project characteristics. Some methodologies, such as OPEN [2], effectively use process patterns as reusable process components for assembling custom processes. Sets of process patterns have also been defined targeting specific development domains; a set of process patterns for agile methodologies was developed in [4], while [5] and [6] provide process patterns for the development of real-time and component-based systems respectively.

Process patterns are now widely used, as process building blocks, in method composition/configuration environments such as Rational Method Composer (RMC) [7] and Eclipse Process Framework Composer (EPFC) [8], which support process engineering by providing state-of-the-art technologies and tools.

B. Aspect-Oriented Processes

During the last decade, the advent of various aspect-oriented processes has resulted in efforts aimed at reviewing and analyzing these processes in order to identify their advantages and deficiencies (e.g. [9, 10]). Consequently, an opportunity has been provided to create generic aspect-oriented processes by exploring the commonalities among existing processes, focusing on the strengths to exploit and the pitfalls to avoid. For instance, as a result of research efforts reported in [11, 12] (among others), generic processes have been proposed for Aspect-Oriented Requirements Engineering (AORE). Furthermore, generic processes have been proposed for architectural AO design [13], and detailed AO design [14]. Most of these processes, however, apply only to specific phases of the development process, and hence miss the lifecycle-wide view that focuses on the continuous chain of development activities and their interrelationships.

Some aspect-oriented processes are proposed as coherent methodologies. Theme [15] is an aspect-oriented methodology spanning the analysis and design phases of the development process with its two main constituents, Theme/Doc and Theme/UML. Aspect-Oriented Software Development with Use Cases (AOSD/UC) [9] is a use-case-driven full-lifecycle methodology. The Aspect-Oriented Component Engineering (AOCE) approach [9] encompasses phases to specify, design, and implement software components by using aspects. Aspect-Oriented Generative Approaches (AOGA) [9] suggests a development process aimed at integrating generative programming and AOSD, encompassing the domain analysis, architectural design, and implementation phases. A combinative approach proposed in [16] is yet another aspect-oriented process which integrates the Theme/Doc approach for the requirements analysis phase, Component and Aspect Model (CAM) [16] for the architectural design phase, and Theme/UML for the detailed design phase.

Some of these methodologies, however, depend on specific technologies, preventing their consideration as generic processes. AOCE, for instance, is appropriate for projects utilizing a component-based development approach. Theme and the combinative approach of [16], although rather general, lack support for important phases such as requirements elicitation and implementation, thus losing comprehensiveness.

No process patterns have been proposed in the domain of AOSD to capture common and well-founded aspect-oriented practices, thus hindering method engineering approaches in targeting this domain. Some method fragments have been proposed in [17], targeted at Aspect-Oriented Modeling (AOM) processes, in order to be used as components in the OPEN methodology's repository. However, it only addresses the *modeling* dimension of aspect-orientation, leaving the remaining process parts (such as architectural design, test, etc.) to conventional methods. In order to fully support SME in an AOSD context, however, there is a need to address other dimensions of aspect-orientation as well, as AOSD manifests itself in different phases, and thus affects the whole development lifecycle.

III. Aspect-Oriented Software Process

In this paper, we propose an Aspect-Oriented Software Process (AOSP), as depicted in Fig. 1, which incorporates and sequences the activities necessary to form a complete AOSD process.

Based upon the AOSP, a set of process patterns have been identified and developed which reflect the significant activities observed in AO processes. The AOSP provides an organization for the process patterns by sequencing them within a generic lifecycle. The patterns, based upon their level of abstraction and granularity, are categorized into the three classes of *phase*, *stage*, and *task*. AOSP contains four serial phase process patterns (Fig. 1), each of them made up of a number of finer-grained iterative stages represented as stage process patterns. The arrow below the diagram represents umbrella activities which span the whole project.

IV. The Proposed Set of Process Patterns for Aspect-Oriented Software Development

This section provides a detailed description of the process patterns comprising the AOSP. AOSP includes general process patterns – referred to as *regular patterns* in this paper – that do not directly apply to aspect-orientation. These are activities that should be included in any typical software process, and are incorporated in AOSP so as to elevate its completeness. Regular patterns and umbrella activities are not further explained in this paper, as they are relatively general, and can hence be adopted from other processes or repositories. We refer to the remaining process patterns as *aspect-oriented (AO) patterns*. These are patterns that introduce processes which either directly deal with crosscutting concerns (e.g. through identification, composition, or conflict resolution between these concerns) or streamline the AO software development endeavor by emphasizing specific conventional development practices (e.g. identification of user requirements and concerns, or techniques of concerns separation).

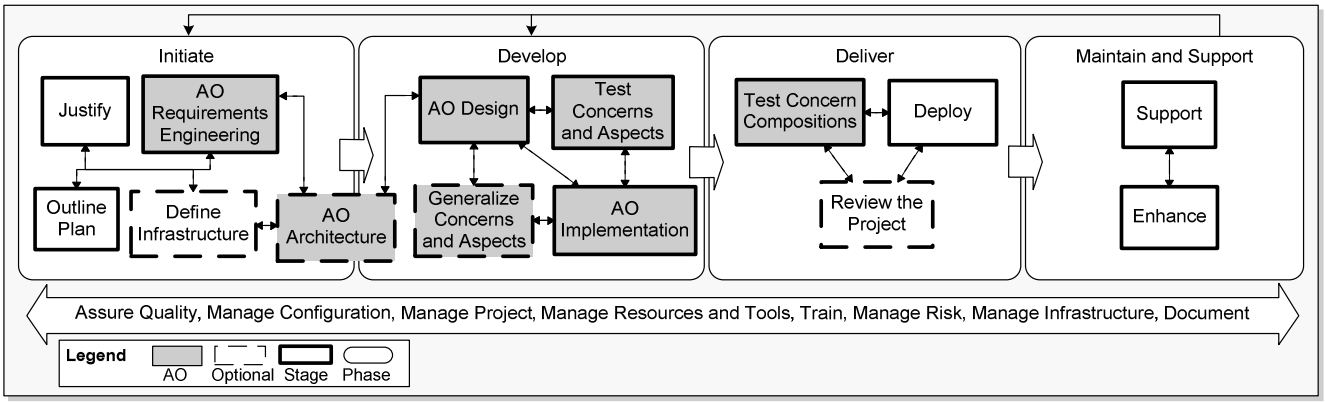

Figure 1. The proposed Aspect-Oriented Software Process (AOSP)

Phase and stage process patterns that contain one or more AO process patterns are also considered as AO patterns. They are shaded in gray in the figures to signify their special importance. Patterns recurring in only a few processes or required in only specific situations are usually considered as *optional*, and are characterized in the figures by dashed borders. Stage process patterns, in contrast, are designated by weighted borderlines within the figures.

A. Phase Process Patterns

The four phases of AOSP include *Initiate*, *Develop*, *Deliver*, and *Maintain and Support* (Fig. 1).

The starting phase of the generic AO process lifecycle is delineated by the *Initiate* phase pattern. The project starts with a preliminary study of the system and a justification for running the project. High-level requirements of the system are identified and defined, and an initial set of crosscutting and non-crosscutting concerns are extracted. Considering system concerns from the very beginning of the project will smooth the transition to the subsequent AO design and implementation. Defining project infrastructure and software architecture, though optional, is critical when facing up to relatively large systems. An initial plan for conducting the project is also outlined in this phase.

The *Develop* phase covers the core development activities, during which the requirements are further detailed and the system is designed and implemented. The design and implementation of the concerns is verified during *Test Concerns and Aspects*. The possibility of generalizing and reusing the concerns is explored during *Generalize Concerns and Aspects*.

In the *Deliver* phase, system-wide testing is performed and the system is deployed to the user environment. A project review is optionally conducted to document the experiences gained.

The *Maintain and Support* phase aims to keep the system running and in production after its deployment to the user environment.

B. Aspect-Oriented Stage Process Patterns

In this section, we describe the activities performed in each of the AO stage process patterns. Due to space limitations, we cannot describe the patterns using a detailed template. Rather, we try to provide a concise informal description which suitably captures the major points of the patterns. For each stage pattern, we mention the pattern name, intent, required/produced artifacts, a short enumeration of its constituent patterns, a diagram, applicable guidelines, and a description of the rationality behind the proposed tasks. Task process patterns are not expanded further; however, whenever applicable, we refer to a number of existing processes as concrete instances that exemplify the task patterns.

A detailed version of the process patterns is being prepared to be published as a method plug-in for the Eclipse Process Framework Composer (EPFC) tool.

1) AO Requirements Engineering: In this stage, system requirements are identified and its high-level features are extracted (Fig. 2). As a result of previous research, generic processes have already been proposed for AO Requirements Engineering (AORE) [11, 12]. This stage pattern has been inspired by these processes, and is somewhat comparable to them. In fact, existing AORE processes have been investigated and encapsulated in the form of process patterns; the AO Requirements Engineering process pattern thus formed has then been refined based on its relationships with other process patterns.

The artifacts input to this stage are: *Business Case*, *Maintenance Plan*, and *Previous Projects Experiences*.

The requirements are extracted during *Analyze Problem/Solution Domain*, which involves *Interviewing Customer*.

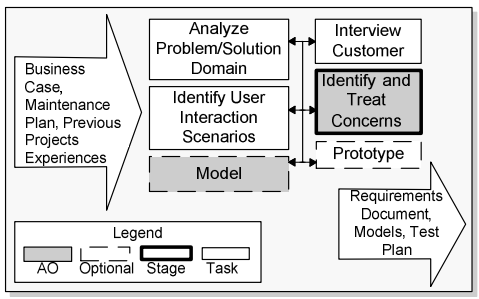

Figure 2. AO Requirements Engineering process pattern

Identifying Interaction Scenarios of different stakeholders with the system will help determine major user concerns. The interaction scenarios and the domain may optionally be modeled. Initial *Test Plans* are also created based on the interaction scenarios.

Identification of the initial concerns of the system is carried out through *Identify and Treat Concerns*, continued until the expected system functionality is satisfied by the set of composed concerns (see the following section). Prototyping may be performed in order to better understand the stakeholders' needs.

The artifacts produced in this stage are: *Requirements Document*, *Models*, and *Test Plan*.

2) Identify and Treat Concerns: During this pattern, system concerns and aspects are identified and handled (Fig. 3). This is a *generic* pattern – performed during requirements analysis, architecture definition, and detailed design – which addresses concerns at different levels of abstraction.

This stage uses the *Requirements Document* and *Existing Models* as input artifacts, which are received from *AO Requirements Engineering*, *AO Architecture*, or *AO Design* process patterns. We start by analyzing the input *Requirements Document* to *Identify and Specify New Concerns*. To efficiently separate and modularize the concerns, each of them are first defined independently of the others, ignoring the effects other concerns may have on it.

Although many AO approaches follow asymmetric separation of concerns [9], the symmetric approach is recommended in this pattern due to the advantages it is believed to bring about. Improved understandability, evolvability, and reusability, for example, are the results of applying the symmetric approach [10].

The relationships between concerns are then explored to detect the impact of each concern on the others. This helps determine the type of the concerns (non-crosscutting or crosscutting), as the crosscutting concerns, or aspects, are related to and usually affect several other concerns. The concerns and the relationships between them are specified and optionally modeled in this pattern.

The Concern-Oriented Requirements Engineering (CORE) approach [12], Theme/Doc [15], and AOSD/UC [9] are instances of this task.

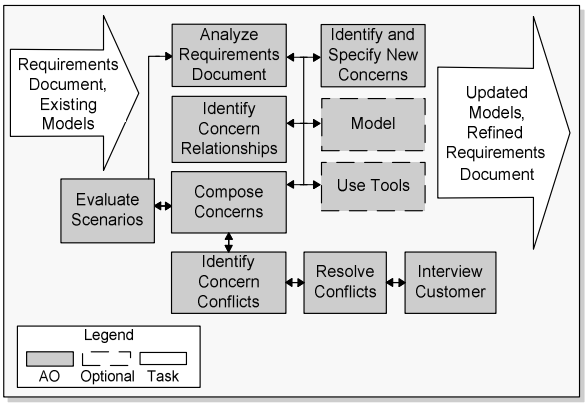

Figure 3. Identify and Treat Concerns process pattern

Theme/Doc, for example, utilizes a visual graph representing the relationships between concerns and requirements to identify candidate aspects, and provides some heuristic questions for sifting through them. The CORE approach, on the other hand, makes use of a relationship matrix to detect which concerns crosscut other stakeholders' requirements in order to identify candidate aspects. It uses a well-defined XML-based template to specify the concerns and their relationships.

Concerns identified thus far are then composed to produce a holistic representation of the system. The composition may be carried out by means of defining and using composition rules to indicate which concerns are to be integrated, and how. Concern composition assists in detecting possible conflicts between the user concerns. The conflicts are then resolved by prioritizing the concerns through negotiation with the software users *(Interview Customer*). Consequently, it may become necessary to revise specific concerns.

Concern composition also allows the evaluation of system integrity (with the help of the interaction scenarios identified in *AO Requirements Engineering*) to ensure that it behaves as expected. If necessary, we can switch back to the earlier tasks of the stage, as a result of which extra concerns may be introduced and/or existing ones may be modified. The processes reported in [12, 18] are instances of this task.

The relationships, conflicts and resolutions, and the composition of the concerns are specified in a composition specification document, as part of output *Models*. The decisions made to resolve the conflicts, the alternative solutions, and the motivations for prioritizing concerns must be thoroughly documented.

MRAT [18] and Theme [15], provide tools to automate the identification and composition of the concerns and aspects, while [19] introduces a tool suite to cover various AORE tasks. The artifacts produced in this stage are: *Updated Models* and refined *Requirements Document*.

3) AO Architecture: Software architectures are known to be good means for addressing software quality issues. The integrated process for AO architectural design proposed in [13] has inspired us in identifying the tasks of this stage (Fig. 4). This stage overlaps with the *Develop* phase, as the architecture prepared in this stage may be refined when elaborating the system during development.

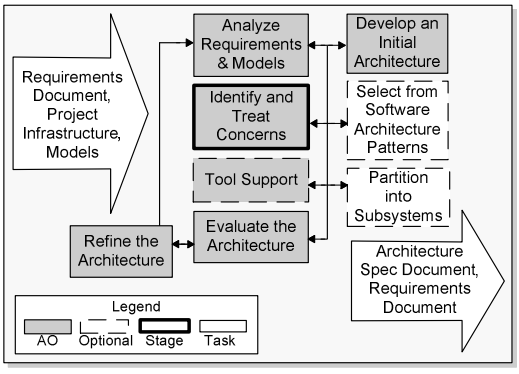

Figure 4. AO Architecture process pattern

The *Requirements Document*, *Project Infrastructure*, and *Models* are the inputs to this stage. System requirements and domain models are first analyzed to develop and model an initial architecture. The pattern *Identify and Treat Concerns* is performed (this time at the architecture level) to identify and extract *architectural concerns* from the input requirements (including the already identified *requirements-level concerns*) which are then added to the aspect-oriented architecture in an iterative-incremental manner.

The scenarios of user interaction with the system are also utilized to discover the architectural concerns/aspects through inspecting the relationships between the scenarios and architectural components.

Some system concerns (e.g. response-time) may not fit into individual architectural aspects, but lead to decisions that affect architectural design. Decisions are also made to address future system changes in order to enhance scalability. All such decisions, along with their driving factors and the alternatives, must be carefully documented. Conventional software architecture patterns can also be utilized, by applying aspectual refactoring wherever necessary. Partitioning a large system into subsystems helps manage its complexity.

To ensure proper modularization, the architecture is evaluated by first looking for scattered concerns and tangled components, through inspecting the relationships between the components and the concerns/scenarios. The architecture is consequently redesigned and refactored, if necessary, by extracting architectural aspects from the scattered concerns/scenarios. COSAAM [20], for instance, is an AO architecture evaluation process which uses Dependency Structure Matrices (DSMs) to derive architectural concerns from scenarios, and provides patterns and heuristics to characterize the modularity of concerns and modules, as well as transformation rules to support the refactoring of the architecture. AO architecture definition and evaluation can be facilitated through the support of tools. DAOP-ADL, AspectLEDA, ASAAM, and PRISMA [21] are instances of architectural design approaches that provide tools for this purpose. The outputs of this stage are: *Architecture Specification Document* and refined *Requirements Document*.

4) AO Design: In this stage we provide the necessary details to implement the software solution (Fig. 5).

This stage accepts the *Requirements Document*, *Models*, *Architecture Specification Document*, *Project Plan*, and *Project Infrastructure* as input artifacts.

While adding low-level details, requirements are elaborated and refined. Design-level concerns are extracted from the refined requirements and handled through *Identify and Treat Concerns,* during which the typically-optional *Model* task is compulsory. We suggest using the Unified Modeling Language (UML) [22] for modeling, since it is employed by nearly all existing AO design approaches [9, 10]. Structural models are utilized to describe concerns and their compositions, and behavioral models are used to describe the interactions required to realize system functions.

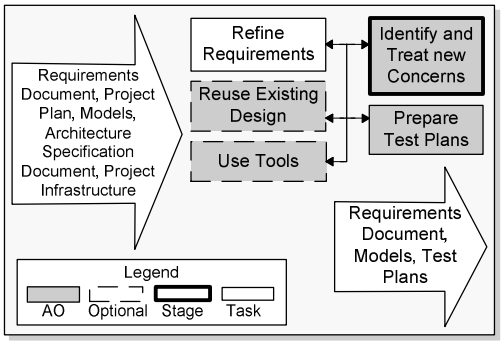

Figure 5. AO Design process pattern

A previously designed artifact/concern may also be reused if it possesses the desired level of reusability, i.e. through the application of the *Generalize Concerns and Aspects* process pattern. *Test Plans* are created for the concerns/compositions in order to prepare for later testing. Theme/UML [15], for instance, presents an AO design process that uses *themes* for encapsulating concerns, which can be used at different levels of abstraction; it also provides a tool for designing compositions based on composition relationships. It is worth mentioning that the generic AO design process proposed in [14] can be inferred from the process patterns of AOSP, such as *AO Design*, *Identify and Treat Concerns*, *Test Concerns and Aspects*, *Test Concern Compositions*, and *Generalize Concerns and Aspects*.

The refined *Requirements Document*, *Models,* and *Test Plans* are the results of this stage.

5) Generalize Concerns and Aspects: During this optional stage, we review the designed concerns to check the possibility for generalizing them so that they can be reused in other contexts.

Potential concerns for reuse are investigated by holding *Generalization Sessions,* during which software artifacts are reviewed and refactored if necessary (Fig. 6). By decoupling concerns, especially the crosscutting ones, from the concrete contexts (*De-contextualize Concerns*), we increase their reusability. The AAM and Theme/UML modeling approaches, for example, use parameterized templates to describe crosscutting concerns [10]. These concerns can therefore be reused in different contexts by binding the template parameters with concrete elements and conditions.

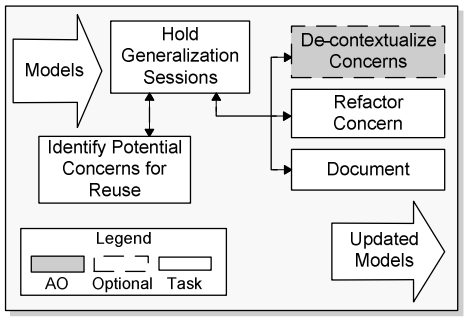

Figure 6. Generalize Concerns and Aspects process pattern

TABLE I. SHORT DEFINITION OF THE *REGULAR* STAGE PROCESS PATTERNS

Pattern	Intent	Required Artifacts	Produced Artifacts
Justify	Justify the project by conducting a feasibility study	Project Description, Previous Projects Experiences	Business Case
Outline Plan	Outline a preliminary plan and schedule for the project; the initial Test Plans are also scheduled and refined in this stage based on the Project Plan.	Requirements Document, Project Infrastructure, Models, Previous Project Experiences, Test Plans	Project Plan, Management Document, Updated Test Plans
Deploy	Deploy the software product to the user environment	Project Infrastructure, Packaged Application, Models	Deployed System, User Documents
Review the Project	Document the project experiences for use in future projects	Plans, Management Document, Project Infrastructure, Models	Project Review Document
Support	Keep the system running and in production	User Request	The user request and solution
Enhance	Respond to the requests for changing the software; changes are usually made as a result of users' feedback during Support.	Change Request, Requirements Document, Models, Project Plan	Upgraded System, Maintenance Plan
Define Infra-structure	Specify the project constraints and standards, and tailor the software process to fit the project at hand; an AO or non-AO programming language is also selected during this stage, as well as tools for automating the tasks of different phases.	Project Description, Requirements Document, Business Case, Models, Previous Projects Experiences	Project Infrastructure document, Updated Models and Requirements Document

6) AO Implementation: In this stage, source code is written based on the *Models*, and is integrated with existing packages (Fig. 7).

The environment chosen in the *Define Infrastructure* stage is used for implementation (Section C). Although not mandatory, it is highly preferred to choose an AO language or environment. If a non-AO language is employed, only the composed models can be used for implementation rather than the models separated based on concerns/aspects; direct mapping between individual concerns and programming constructs is thus lost. Bugs identified during tests are also corrected in this stage. *Source Code* and *Packaged Application* are the outputs of this stage.

7) Test Concerns and Aspects: In this stage, we perform tests at the level of individual artifacts and concerns.

This stage accepts *Models, Source Code, Requirements Document*, and *Test Plans* as input artifacts (Fig. 8). *Unit Testing* and *Model and Code Walkthrough* are typical tasks performed during this stage. Test cases are created and run for concerns and aspects. Whenever the system is changed, e.g. a new concern is added to the system or an existing one is modified, we perform *Regression Testing*. *Tested Artifacts* and *Test Results* are the outputs of this stage.

8) Test Concern Compositions: In this stage, high-level tests are performed on the entire system (Fig. 9). Activities of this stage are similar to the ones in *Test Concerns and Aspects*, yet tests are designed at the system-level.

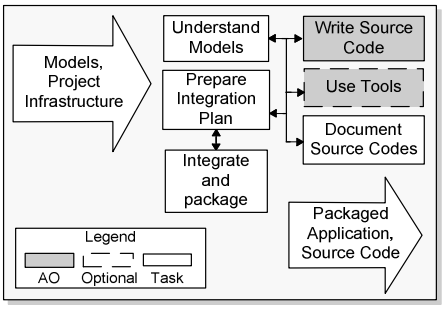

Figure 7. AO Implementation process pattern

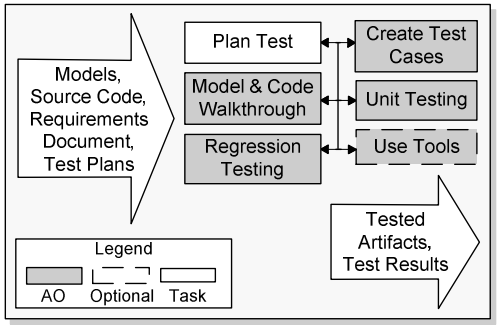

Figure 8. Test Concerns and Aspects process pattern

The user interaction scenarios identified during requirements engineering are utilized to create high-level test cases. Behavioral test cases can be derived from AO behavioral models, as proposed in [23, 24, 25]. Whenever a new concern is added to the system, *Regression Testing* is performed as well as *Integration Testing* on the concern composition specifications. Defects thus discovered are fixed through *Fix Bugs*.

C. Regular Stage Process Patterns

A short definition for the *regular* patterns is provided in Table I. Detailed definitions are provided in [1].

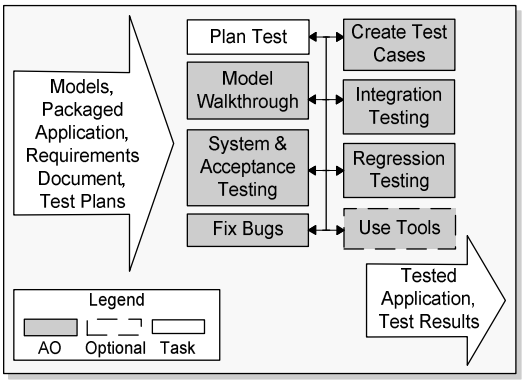

Figure 9. Test Concern Compositions process pattern

V. CRITERIA-BASED ANALYSIS OF THE PATTERNS

Many approaches and tools have been proposed for assessing AO software *artifacts*, e.g. models, documents, and source code [10, 26, 27]. However, there are few full-fledged mature methods that provide a criteria-based evaluation framework for AO software *processes*. The proposed analysis criteria (e.g., [9, 10, 28, 29]) are either only pertinent to specific phases of development, or lack the desired maturity.

We therefore decided to introduce our own evaluation framework, which consists of coherent sets of analysis criteria for assessing AO processes as well as process patterns. The criteria have been compiled through an iterative refinement process: A collection of analysis criteria was initially prepared by studying various resources; the collection was then iteratively refined in accordance with certain validity *meta-criteria* (criteria used to evaluate other criteria). After the criteria were fixed, the proposed process patterns were analyzed through applying the criteria.

To provide a fair set of evaluation criteria, we first took into account the meta-criteria cited in [30], including A) *Generality of criteria*, in order for the criteria to be applicable to a wide range of methods, B) *Preciseness*, to effectively distinguish similarities and differences between methods by using the criteria, C) *Comprehensiveness*, so as to enhance the coverage of main aspects of methods by the criteria, and D) *Balance of criteria*, to address the *technical*, *managerial*, and *usage* issues in methods.

We have, for example, investigated different resources on the evaluation of both AO and non-AO processes to make the criteria comprehensive as well as balanced. Consequently, three additional meta-criteria were defined to address the issue, listed as follows:

I) Inclusion of *general* criteria expected from any typical methodology;
II) Inclusion of *aspect-oriented* criteria to address AO-related issues;
III) Consideration of criteria to assess *process patterns* in general.

We then checked the set of criteria for possible incompatibilities and overlaps, while trying to keep the set as small and effective as possible. In order to address generality, we considered those criteria which were applicable to a wide range of process types.

For assessing a software process from a *general* perspective (meta-criterion I) a number of resources [31, 2, 32] were used as the main sources for forming our evaluation framework. The criteria obtained from these sources were subsequently pruned so as to include only the items appropriate for AOSD processes. For evaluating *AO processes*, on the other hand, a number of methods have been proposed as well [9, 10, 28, 29]. However, preparation of the initial evaluation criteria (concerning meta-criterion II), was most influenced by the extensive analytical survey reported in [9].

No specific approach has been proposed in the literature for criteria-based evaluation of *process patterns*. We therefore resorted again to general evaluation criteria and tailored them by following the meta-criteria of [30] to meet pattern evaluation requirements. Expert advice was extensively used in distilling and refining the criteria.

For quantifying the evaluation results, a method similar to the *Feature Analysis* approach was followed [33]. As the approach suggests, a list of relevant features to be assessed is produced, which is then rated by an individual or a group according to a predetermined rate scale.

TABLE II. GENERAL ANALYSIS CRITERIA FOR SOFTWARE PROCESSES

Criterion	Type / Definition		Value / How realized
Coverage of Generic Lifecycle	E	The phases of the generic software development lifecycle that are covered by the process.	Requirements Engineering, Architecture, Design, Implementation, Test, Deployment, and Maintenance
Support for Umbrella Activities	S	A: No support for umbrella activities B: Supported, yet leaving the concrete definition of the activities to the developer/method-engineer C: Supported by providing specific methods for umbrella activities	B
Configurability/ Extensibility/ Flexibility/ Scalability	D	The means by which these criteria are satisfied in order for the process to fit different project situations.	Naturally satisfied by the concept of process patterns [31]. Also supported by the *Define Infrastructure*, *AO Architecture*, *Model*, and *Generalize Concerns and Aspects* process patterns, as well as the tool support currently available.
Application Scope	D	The domains for which the process is applicable.	General, as the AO approach is based upon object-orientation.
Traceability Throughout Lifecycle	D	Support for traceability between different states of artifacts/concerns through the lifecycle.	Supported by separating concerns along with their corresponding models, and following an iterative-incremental approach to development.
Verification and Validation	D	The means by which stakeholders are able to verify and validate the requirements, and also to validate intermediate artifacts/decisions against the requirements.	Supported through identification of user interaction scenarios, used in subsequent evaluations, and also *Test Concerns and Aspects* and *Test Concern Compositions*
Tool Support	D	Whether the process is supported by specific tools.	*For AO processes:* Partially supported by suggesting relevant external tools. *For Process Patterns:* Supported by existing process authoring/configuration environments [7, 8].
Reusability of Artifacts	D	The ease to reuse process artifacts for different projects [28].	Through *Generalize Concerns and Aspects* process pattern.

TABLE III. ANALYSIS CRITERIA FOR ASPECT-ORIENTED PROCESSES

Criterion	Type / Definition		Value / How realized
Concern Identification and Treatment	S	A: No support for identifying (crosscutting) concerns B: Ability to identify and handle crosscutting concerns C: Ability to identify and handle both crosscutting and non-functional concerns	C
Composability	S	A: No support for composition B: Syntactic support for composition C: Syntactic support for composition as well as semantic support to comprehend the composition and to identify conflicts	C (Supported through the tasks of *Identify and Treat Concerns* process pattern)
Trade-Off Analysis	P	Trade-off analysis and resolution for concern conflicts	Supported
Compositional Separation	E	The level of separation – symmetric or asymmetric – of concerns	Adaptable to both (yet with tendency towards symmetric separation)
Evolvability	D	The ability to add, remove or change individual artifacts – concerns, requirements, and the relevant models – with ease (aka Change Propagation).	Supported through (symmetrically) separating the concerns and maintaining their relation with the corresponding artifacts/decisions, and also through the use of tools
Support for Mapping	P	Support of the process for mapping concerns – especially the crosscutting ones – to architectural, design, or implementation decisions	Partially supported though documenting the relationships and dependencies between the concerns and the decisions made to realize them
Alignment to Phases	S	Alignment of the concepts of the approach to requirement-level concerns and/or implementation-level concepts/constructs: A: To none, B: To either requirements or implementation, C: To both	Aligned to both
Homogeneity of Treatment	P	Homogeneity of treatment process for different types of requirements/concerns (functional or non-functional, crosscutting or non-crosscutting)	Supported
Platform Independence	P	Whether the concepts of the approach are independent from (unaffected by) any specific platform or programming language	Supported

Following the *Feature Analysis* approach and aiming to define our criteria as precisely as possible, we presented them in four types of forms: A three-level *Scale Form* (S), with a short definition for each level, indicating the degree to which a particular feature of the method is satisfied; an *Enumerated Form* (E), with a list of possible values for the criterion; a *Simple Form* (P), with a "yes/no" answer to whether or not a particular feature is supported by the method; and a *Descriptive Form* (D) – for criteria that could hardly be graded according to a fixed set of degrees – which describes the way a criterion is satisfied and the rationality behind it in a narrative form, which should be as clear as possible to avoid subjective evaluations.

The evaluation criteria were ultimately refined into three categories, which respectively correspond to the meta-criteria I, II, and III. The criteria, along with a short definition for each, the type and range of the domain values, and the ratio to which they are realized by the proposed AOSP and process patterns (evaluation results), are presented in Tables II, III, and IV, corresponding to the three categories respectively.

TABLE IV. ANALYSIS CRITERIA FOR PROCESS PATTERNS

Criterion	Type / Definition		Value / How realized
Template Formality	S	A: No predetermined template B: Conformance to a concise semi-formal/informal template C: Conformance to a detailed and well-structured formal template	B
Consistency	S	Consistency amongst patterns, in terms of input/output work products within a pattern (local consistency) and between different patterns (global consistency): A: No consistency B: Support for either local or global consistency C: Support for both local and global consistency	C
Complexity Management	P	Provision of techniques to manage large numbers of patterns and/or to manage large patterns	Supported (By categorizing the patterns into phases, stages, and tasks)
Determination of Work Products	P	Determining which work products are involved in each process pattern	Supported
Determination of Roles	P	Determining which roles are involved in each process pattern	Not supported (work in progress)
Classification of Work Products/Roles	P	Proposal of a classification scheme for work products/roles	Not supported (work in progress)
Cohesion	E	The levels of cohesion satisfied by process patterns	Functional, sequential, procedural, and temporal cohesions
Coupling	E	The levels of coupling that exist between process patterns	Data coupling
Instantiation Guidance	P	Offering techniques/guidelines for instantiation/composition of process patterns	Not supported (work in progress)
Existence of Configurations of Process Patterns	P	Whether there exist any empirical or illustrative configurations of process patterns (explicitly or implicitly) regarding specific project situations, to exemplify the practicality of the application and instantiation of process patterns	Not supported (work in progress)

VI. Conclusions and Future Work

The proposed generic aspect-oriented software process (AOSP) reflects the high-level activities considered necessary to form a complete AO software development process, and can serve as a framework for instantiating AO methodologies. The proposed process patterns are extracted from renowned and/or successful AO processes, and represent prominent processes and practices for different stages of an AO software development lifecycle. They can well serve as method fragments in an SME repository for assembly-based engineering of AOSD methodologies.

AOSP outlines a generic lifecycle for AO methodologies. Even though it is abstract and generic, it organizes the AO practices that can be applied in software development, and provides a synergistic organization for the relevant process patterns. To gain a better understanding of the synergy observable in AOSP, consider the following examples: the *AO Requirements Engineering* process pattern identifies the concerns (non-crosscutting and crosscutting) and system usage scenarios which will later help in the identification of architectural- and detailed-design concerns and aspects, as well as in the evaluation of subsequent artifacts; module-level AO testing is performed in the core development phase, whereas integration and system-wide AO testing is deferred to the end of development iterations and the delivery phase; moreover, AOSP carefully defines the proper flow of AO artifacts among process patterns. In other words, AOSP is larger than the sum of its parts, mainly because it defines and governs the static and dynamic relationships and interactions that should exist among its constituent process patterns.

Work is currently underway to detail the task process patterns in order to publish them to the Eclipse Process Framework Composer (EPFC) [8] environment. In order to empirically validate the AOSP and the process patterns, they will be used for assembly-based construction of concrete processes to fit specific project situations. Reifying the process patterns in real projects can also help refine the patterns and enhance their practicality.

Acknowledgment

We wish to thank Mr. Yusef Mehrdad for assisting in the refinement of the evaluation criteria.

References

[1] Ambler, S. W., Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press, 1998.

[2] R. Ramsin, and R. F. Paige, "Process-centered review of object oriented software development methodologies," ACM Computing Surveys, vol. 40, no. 1, Feb. 2008, pp. 1-89.

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

[4] S. Tasharofi, and R. Ramsin, "Process patterns for Agile methodologies," Situational Method Engineering: Fundamentals and Experiences. J. Ralyté, S. Brinkkemper, B. Henderson-Sellers, Eds., Springer, 2007, pp. 222-237.

[5] N. Esfahani, S. H. Mirian-Hosseinabadi, and K. Rafati, "Real-time analysis process patterns," Proc. CSICC'08, Mar. 2008, pp. 777-781.

[6] E. Kouroshfar, H. Yaghoubi Shahir, and R. Ramsin, "Process patterns for component-based software development," Proc. CBSE'09, Jun. 2009, pp. 54-68.

[7] P. Kroll, "Introducing IBM Rational Method Composer", 2005, Published on the web at: http://www.ibm.com/developerworks/rational/library/nov05/kroll.

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

[9] R. Chitchyan, et al., "Survey of analysis and design approaches," AOSD-Europe, Deliverable D11, 2005.

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

[11] R. Chitchyan, A. Sampaio, A. Rashid, P. Sawyer, and S. S. Khan, "Initial version of aspect-oriented requirements engineering model," AOSD-Europe, D36, 2006.

[12] A. Moreira, J. Araujo, and A. Rashid, "A concern-oriented requirements engineering model," Proc. CAiSE'05, LNCS 3520, Jun. 2005, pp. 293-308.

[13] I. Krechetov, B. Tekinerdogan, M. Pinto, and L. Fuentes, "Initial version of aspect-oriented architecture design approach," AOSD-Europe, D37, 2006.

[14] A. Jackson, and S. Clarke, "Towards a generic aspect-oriented design process," Proc. Int'l Workshop on Aspect-Oriented Modeling, MoDELS'05, 2005, pp. 110-119.

[15] E. Baniassad, and S. Clarke, "Theme: an approach for aspect-oriented analysis and design," Proc. ICSE'04, May 2004, pp. 158-167.

[16] P. Sánchez, L. Fuentes, A. Jackson, and S. Clarke, "Aspects at the right time," TAOSD IV, vol. 4640, LNCS, Nov. 2007, pp. 54–113.

[17] B. Henderson-Sellers, R. France, G. Georg, and R. Reddy, "A method engineering approach to developing aspect-oriented modelling processes based on the OPEN process framework," Information and Software Technology, vol. 49, no. 7, Jul. 2007, pp. 761-773.

[18] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters, "Semantics-based composition for aspect-oriented requirements engineering," Proc. AOSD'07, Mar. 2007, pp. 36-48.

[19] R. Chitchyan, A. Sampaio, A. Rashid, and P. Rayson, "A tool suite for aspect-oriented requirements engineering," Proc. Int'l Workshop on Early Aspects, ICSE'06, ACM, May 2006, pp. 19-26.

[20] B. Tekinerdogan, F. Scholten, C.Hofmann, and M. Aksit, "Concern-oriented analysis and refactoring of software architectures using dependency structure matrices," Proc. Workshop on Early Aspects, AOSD'09, Sep. 2009, pp. 13-18.

[21] A. Navasa, M. A. Pérez-Toledano, J. M. Murillo, "An ADL dealing with aspects at software architecture stage," Information and Software Technology, vol. 51, no. 2, Elsevier, Feb. 2009, pp. 306-324.

[22] OMG, "Unified Modeling Language Specification, Version 2.0," Technical Report, OMG, 2005.

[23] A. Jackson, J. Klein, B. Baudry, S. Clarke, "KerTheme: testing aspect oriented models," Proc. IMDT workshop at ECMDA'06, 2006.

[24] P. Massicotte, L. Badri, M. Badri, "Towards a tool supporting integration testing of aspect-oriented programs," Journal of Object Technology, vol. 6, no. 1, Jan.-Feb. 2007, pp. 67-89.

[25] W. Xu, and D. Xu, "A model-based approach to test generation for aspect-oriented programs," Proc. Workshop on Testing Aspect-Oriented Programs, AOSD'05, 2005.

[26] E. Figueiredo, A. Garcia, C. Sant'Anna, U. Kulesza, and C. Lucena, "Assessing aspect-oriented artifacts: towards a tool-supported quantitative method," Proc. Workshop on QAOOSE, ECOOP'05, 2005.

[27] P. Greenwood et al., "On the contributions of an end-to-end AOSD testbed," Proc. Workshop on Early Aspects'07 at ICSE, IEEE Computer Society, May 2007.

[28] G. S. Blair, L. Blair, A. Rashid, A. Moreira, J. Ara´ujo, and R. Chitchyan, "Engineering aspect-oriented systems," Aspect-Oriented Software Development. R.E. Filman, T. Elrad, S. Clarke, and M. Aksit, Eds., Addison-Wesley, Oct. 2005, pp 379-406.

[29] S. S. Khan, and M. J. Rehman, "A survey on early separation of concerns," Proc. APSEC'05, IEEE Computer Society, Dec. 2005, pp. 776-782.

[30] G. M. Karam, and R. S. Casselman, "A cataloging framework for software development methods," Computer, vol. 26, no. 2, Feb. 1993, pp. 34-45.

[31] R. Ramsin, "The engineering of an object-oriented software development methodology," Ph.D. Thesis, University of York, UK, 2006.

[32] M. Taromirad, and R. Ramsin, "An appraisal of existing evaluation frameworks for Agile methodologies," Proc. ECBS'08, IEEE Computer Society, 2008, pp. 418-427.

[33] K. H. Fung, G. C. Low, "Methodology evaluation framework for dynamic evolution in composition-based distributed applications," Journal of Systems and Software, vol. 82, no. 12, Elsevier, Dec. 2009, pp. 1950-1965.