Patterns of Situational Method Engineering

Mohsen Asadi and Raman Ramsin

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

Summary. Situational Method Engineering (SME) addresses the need for custom-built software development methodologies that are tailored to fit specific project situations. A *process pattern* is a description of a recurring development process fragment that can be used as a generic model or a building block for engineering development processes. *Software* development process patterns are extensively used in SME, mainly as process components assembled to form bespoke methodologies; however, the SME field itself has not been scrutinized as to the *methodology* development process fragments frequently encountered. Situational method engineering knowledge captured in the form of SME process patterns is not only useful for building and improving SME processes, but can also facilitate knowledge transfer among method engineers. We propose a set of SME process patterns obtained through studying existing method engineering approaches. The set of patterns is organized into a generic pattern-based framework for SME. The framework can be used for developing SME processes according to the specific requirements of method engineering projects; the resulting SME processes can then be enacted to yield custom-built, project-specific methodologies.

1 Introduction

A Software Development Methodology (SDM) is a framework for applying software engineering practices with the specific aim of producing software-intensive systems [1]. SDMs and supporting tools have become key factors in achieving success in software development projects. Traditional, rigid SDMs are inadequate for providing the necessary support for modern information systems development projects. It has therefore become necessary to develop custom-built methodologies.

Method Engineering (ME) strives to improve the usefulness of SDMs, mainly through using adaptive frameworks for constructing and/or adapting SDMs. The most well-known subfield of the discipline is *Situational* Method Engineering (SME), which is concerned with the construction or adaptation of a methodology according to the characteristics of the project situation at hand [2]. Existing SME approaches are classified as: *Ad-hoc* [3], in which a new methodology is constructed from scratch; *Paradigm-based* [4], in which an existing meta-model is instantiated, abstracted and/or adapted in order to produce the target methodology; *Method Configuration (adaptation/extension)* [5], which aims at enhancing a base methodology by adding/removing elements and features; *Assembly-based* [6], in which method engineers construct the target methodology or enhance an existing methodology through reusing

R. Lee & N. Ishii (Eds.): Soft. Eng. Research, Manage. & Appli. 2009, SCI 253, pp. 277–291.
springerlink.com

process components; *Hybrid Methodology Design* [1, 3], in which the target methodology is built through a top-down iterative-incremental process which combines different approaches; and *Formal Agile Method Engineering* [7], in which a methodology is created through applying policies similar to those seen in agile software development approaches.

A *Process Pattern* defines a general solution to a certain recurring problem in software development processes [3, 8]. Process patterns can be considered as reusable process components, as seen in certain implementations of assembly-based SME, where software engineering processes are constructed through reusing software engineering process patterns [6]. Software engineering has thus benefited much from process patterns, and as a result, prominent heavyweight SDMs have been replaced or enhanced by assembly-based method engineering frameworks; for instance, OPEN has become OPF, and RUP has been fused into RMC [1]. We intend to bring these advantages to the situational method engineering domain itself, mainly through defining a process-pattern-based approach for constructing SME processes.

We propose a pattern-based framework as a meta-model for defining SME processes. A generic process model for SME has been created through instantiating this framework and populating it with several process patterns that we have extracted from existing SME approaches. The generic process model can be specialized to produce bespoke SME processes, mainly through setting the precise order of the SME activities involved. It thus allows combining existing SME approaches – manifest in its constituent components – in various lifecycle styles (e.g., iterative-incremental).

This paper is organized as follows: The pattern-based framework is described in Section 2; Section 3 describes the different creation policies and construction blocks used in SME approaches; Section 4 defines the *Situational Method Engineering Process (SMEP)* as an instance of the proposed framework; the process patterns comprising the SMEP are described in Section 5; Section 6 provides an example of applying SMEP for engineering a real-time systems development methodology; and Section 7 contains the conclusions and some suggestions for furthering this research.

2 Proposed Pattern-Based SME Framework

We define and use the process framework shown in Fig. 1 as a meta-model for pattern-based SME processes; the framework is based on the process-pattern-based approach proposed in [9]. *Process patterns* are the core elements of the framework, representing strategies for solving certain recurring SME *problems*. A *problem* is a concrete situation that may arise during methodology development. In the method engineering domain, a variety of problems can exist at different levels of granularity. Based on the granularity level of the problem addressed, we can define three different types of process patterns; namely, *task*, *stage*, and *phase*. A *task* process pattern depicts detailed steps performed to solve a specific fine-grained problem of the SME process. A *stage* process pattern defines steps that need to be executed in order to solve a stage problem of the SME process, and is usually made up of several task process patterns. Finally, a *phase* process pattern represents the interaction of two or more stage process patterns to solve the phase problem to which they belong [8, 10].

Each process pattern needs an *initial context* in order to produce a *result context*. The initial context describes a project situation in which we can apply a process pattern. The result context describes the situation after applying the pattern. *Work products* are the artifacts created during the development process. Instances of work products are method fragments, product models, and process models. Each process pattern is applied by one or more *roles*. A specific role is assigned to a *person* or *tool*.

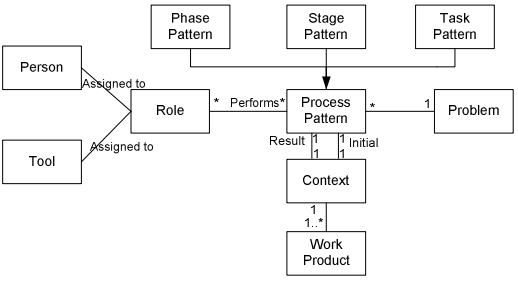


Fig. 1. Pattern-based SME framework

3 Infrastructures and Creation Policies in SME Approaches

SME approaches can be differentiated based on two characteristics: *Infrastructure* and *Creation Policy*. *Infrastructure* shows the kinds of resources that a SME approach uses to create custom methodologies. Four types of infrastructure can be identified: (a) base methodology, (b) meta-model, (c) process component (chunk), and (d) configuration package. A *base methodology* is composed of a *product model* and a *process model* [4]; the product model defines the set of elements (artifacts) produced, their properties, and the relationships that are needed to express the outcome of a process; the process model consists of the set of goals, activities and guidelines required to support the process's intent [4]. *Meta-models* abstract methodology concepts, their inter-relationships, and the constraints binding them together, into a coherent framework [11]. A *method chunk* is an independent and cohesive methodology part, supporting the realization of specific software development activities, and typically stored in a repository of reusable process components [6]. A configuration characteristic defines a problem occurring in a development situation, the solution of which is achieved through configuring a base methodology [6]. The configuration of a base methodology according to a specific characteristic is called a *configuration package* [5]. According to [12], there are various granularity levels for a methodology, represented as five layers: *Method*, *Stage*, *Model*, *Diagram*, and *Concept*. Each of the infrastructures defined above can reside on different granularity layers: method chunks, configuration packages, and meta-models can reside on all of the five layers. The base methodology, however, can only reside on the method layer.

A *creation policy* represents the approach that can be adopted for creating a methodology. Main policies include [3, 6]: *Assembly*, *Abstraction*, *Instantiation*, *Method Configuration, Artifact-Oriented* (devising a seamless complementary chain

of artifacts and building the process around it), and *Integration* (integrating features and techniques taken directly from existing methodologies).

Every SME approach is based on specific infrastructures and creation policies: For instance, the assembly-based approach uses method chunks and adopts the assembly policy; the configuration-based approach uses a methodology and a configuration package and adopts the configuration policy; the hybrid approach uses meta-models, method chunks and a base methodology and adopts the assembly, instantiation, artifact-oriented, and integration policies; and the paradigm-based approach uses a meta-model or a base methodology and adopts the instantiation or abstraction policy.

4 Situational Method Engineering Process (SMEP)

It has been pointed out that software processes are software too [13]; we can therefore utilize a software engineering strategy to create SDMs. We can also consider a lifecycle for method engineering that is similar to the generic software development lifecycle, consisting of the generic phases of: requirements engineering, analysis, design, implementation, test, deployment, and maintenance [3]. Based on this notion, we have instantiated the process-pattern-based framework described in section 2 to define the *Situational Method Engineering Process (SMEP)* as a generic SME process model (Fig. 2). SMEP's constituent process patterns were obtained through studying existing SME approaches. SMEP is made up of three serial phases, which in turn consist of iterative stages. The first phase initiates the methodology and sets the foundation for methodology construction. The methodology is developed and deployed into the execution environment in the next two phases. SMEP uses an iterative-incremental approach for creating software development methodologies. First, an appropriate infrastructure and a suitable policy will be chosen to create each part of a methodology. This selection is based on the level of granularity at which we are implementing the methodology, and the elicited requirements that should be realized during each iteration (analogous to the Hybrid Design approach [3]). For example, during the initial iterations, the instantiation and abstraction policies would be selected to create a base methodology. Then, the assembly policy will be applied to complete the base methodology by putting together existing method chunks. Finally, the configuration-based policy will be used to extend or restrict the new methodology. Since SMEP covers all existing SME approaches, it can be considered a generic process model for SME.

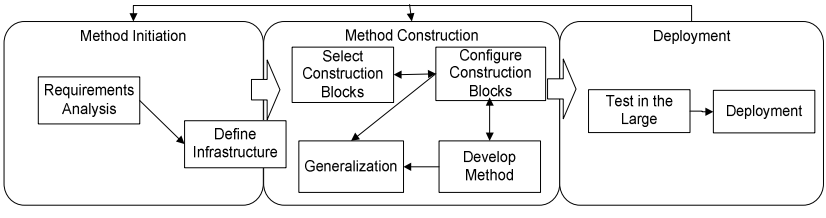


Fig. 2. The proposed Situational Method Engineering Process (SMEP)

5 SMEP Process Patterns

In this section, we describe the constituent process patterns of the SMEP. We use the granularity level, as defined in the pattern-based framework, to categorize the extracted process patterns.

5.1 Phase Process Patterns

SMEP consists of three serial phases: *Method Initiation*, *Method Construction* and *Method Deployment*. Phase process patterns reside in the topmost granularity layer of the framework. They consist of stage process patterns which interact to achieve the objectives of their relevant phase. As seen in section 3, there are various creation policies. Since we can use all of these policies in the SME process, we will consider each creation policy as a workflow in the methodology construction phase.

Method Initiation Phase. The main goal of this phase is to provide a foundation for the successful construction of the target methodology. The *requirements analysis* and *infrastructure setup* stages are performed iteratively during this phase, producing the requirements model, required infrastructure, and methodology architecture.

Method Construction Phase. This phase produces the project-specific methodology in an iterative-incremental manner; each iteration first selects the appropriate workflow, and then determines the tasks and stages in that workflow. *Select construction blocks*, *configure construction block*, *generalization*, and *develop method* are the stages of this phase.

Deployment Phase. The objective of this phase is to successfully put the methodology into production. This also means that the method engineer needs to perform overall testing on the methodology (as part of the *testing in the large* stage), mainly to verify the completeness of the methodology and validate it against the requirements. The methodology is then deployed into the user environment.

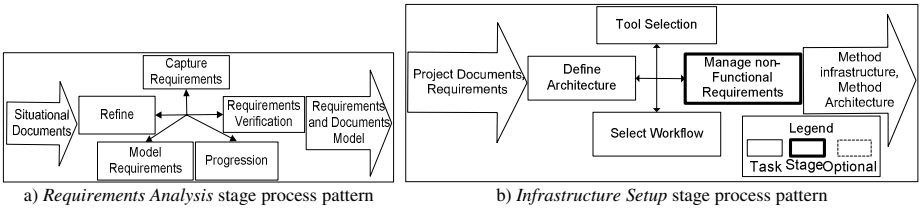
5.2 Stage Process Patterns

Stage process patterns comprise the bulk of each phase. Most stage patterns are executed iteratively. This section explains each stage pattern of the SMEP. The solution, tasks performed to realize the stage objective, and initial/result contexts will be explained for each stage pattern.

Requirements Analysis Stage. The objective of this stage is to define methodology requirements (Fig. 3(a)). It receives specific situation documents as input, which should specify three main attributes: *Definition*, *Domain*, and *Deliverables*; *Definition* explains the type of the project at hand; *Domain* specifies the application domain of the target system; and *Deliverables* describes the artifacts that should be produced [14]. Requirements can be seen from two viewpoints. From the first viewpoint, requirements are divided into two categories: *General Requirements,* such as repeatability, understandability, etc.; and *Situational Requirements*, such as requirements for business applications, real-time applications, etc. The second viewpoint divides the requirements into two classes: *functional requirements,* spanning the features that the methodology should provide, such as work products and required activities; and

non-functional requirements, such as smoothness of transition between activities, robustness, and scalability. The *Capture Requirements* task elicits, documents, and agrees on methodology requirements (functional and non-functional). The *Refine* task details, decomposes, aggregates, and identifies alternatives for the requirements. The *Model Requirements* task creates a model for requirements which can be shown by a special state diagram, in which states represent work-products produced by the methodology, whereas transitions show the milestones and required activities. New requirements that fill the gaps in the requirements set are defined through the *Progression* task. The *Requirements Verification* task finalizes the requirements analysis process by verifying the requirements model's completeness and coherence [15].

Infrastructure Setup Stage. This stage aims to provide the necessary resources (infrastructure and architecture) for creating a methodology. Inputs to this stage include requirements documents, project description, and experiences gained from previous projects (Fig. 3(b)). The *Tool Selection* task identifies the appropriate Computer-Aided Method Engineering (CAME) environments [2] to be used during methodology construction. The *Define Architecture* task provides the backbone of the methodology, defining the main phases of the lifecycle and the main principles that should be followed for software development during methodology enactment. The *Manage Non-Functional Requirements* stage analyzes the non-functional requirements and identifies the relevant factors/tactics according to the methodology architecture. The *Select Workflow* task selects a workflow for the current iteration of method construction according to the requirements and architecture.



a) *Requirements Analysis* stage process pattern b) *Infrastructure Setup* stage process pattern

Fig. 3. Components of *Method Initiation* phase process pattern

Select Construction Blocks Stage. The objective of this stage is to select the construction blocks according to the selected requirements and then perform preliminary adaptation on them. The type**s** of construction blocks selected - i.e. method chunks, meta-model or methodology – depends on the type of workflow selected in the infrastructure stage. This stage receives the requirements, method architecture, and infrastructure as input (Fig. 4).

***Assembly Workflow*:** The *Select Method Chunk* task converts the requirements to appropriate parameters and uses the matching technique to select the chunk according to the requirements. The *Chunk Evaluation* task validates the retrieved method chunks by evaluating the degree of matching between the candidate chunk and the requirements. The *Chunk Decomposition* task selects the relevant sub-chunks and eliminates the inadequate/unneeded ones. The *Chunk Aggregation* task produces an aggregate chunk from the candidate chunks, based on the observation that in many cases, an

aggregate chunk is functionally larger than the sum of its sub-chunks, and might therefore provide a solution for the requirements not addressed by the constituent chunks. The *Chunk Refinement* task refines the candidate chunk by providing an in-depth definition of the chunk according to the requirements.

***Paradigm Workflow*:** The objective of this workflow is to create the product model of the target methodology, to be used in the next stages for creating the corresponding process model. The paradigm model, which can be either a process or a product model/meta-model, is specified and modified by the *Select Paradigm Model* and *Adapt Paradigm Model* tasks. The *Analyze Paradigm Model* task investigates and identifies elements of the paradigm model which should be either abstracted or instantiated. Then, based on the type of the paradigm model (methodology or meta-model), the *Abstraction* or *Instantiation* task is executed. The instantiation and abstraction tasks can be further divided into *Process-based* or *Product-based* tasks based on the type of the paradigm model. If process-based abstraction or instantiation is performed, then the *Identify the Corresponding Work Product* task identifies and integrates the relevant work products, thereby creating the product model.

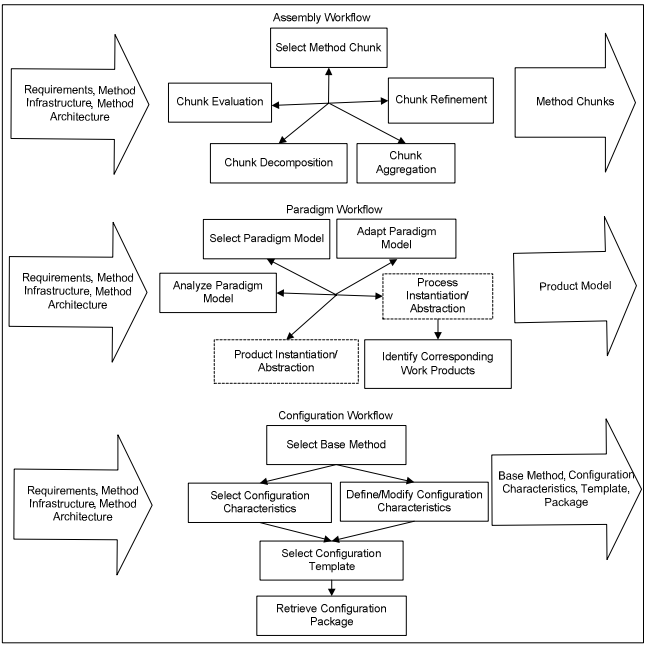
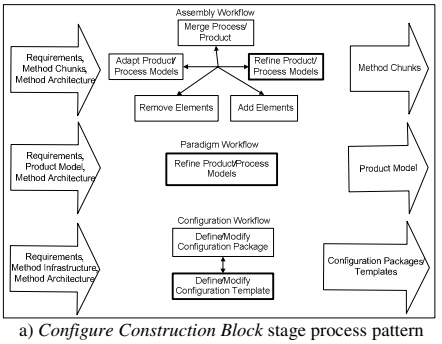
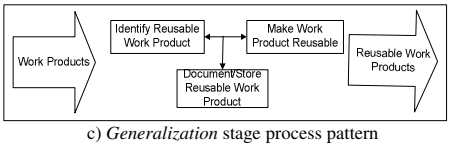


Fig. 4. *Select Construction Blocks* stage process pattern

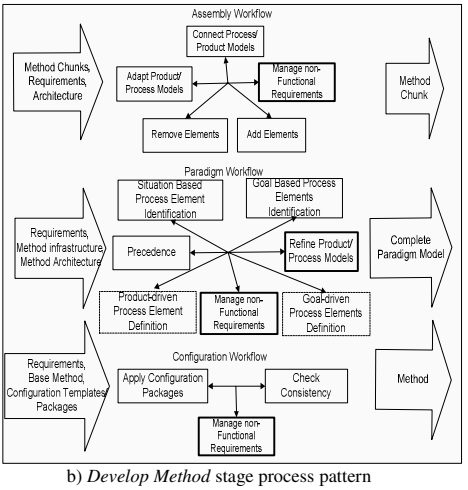
***Configuration workflow*:** The *Select the Base Method* task analyzes the requirements and selects the relevant base methodology. According to the requirements and the selected base methodology, the configuration characteristics of the project are identified by means of analyzing a repository of development situations and characteristics. Configuration templates are then identified and matched to the characteristics. The



a) *Configure Construction Block* stage process pattern



c) *Generalization* stage process pattern



b) *Develop Method* stage process pattern

Fig. 5. Method development stage process patterns

configuration packages corresponding to each configuration template are retrieved from the repository of configuration packages.

Configure Construction Block Stage. This stage performs configuration on the result of the previous stage. Inputs of this stage are the requirements, construction blocks, and method architecture. It provides enhanced construction blocks for creating and/or modifying the target methodology (Fig 5(a)).

***Assembly Workflow*:** Some of the selected chunks may have similar engineering goals while providing different ways to satisfy them. In such cases, the process and product models of the chunks overlap; integration is therefore necessitated. The *Adapt the Product/Process Models* task unifies the names of the similar product/process concepts and performs the required transformations to create the adapted product/process models. After creating the adapted models and removing the inconsistencies, the *Merge Process/Product* task combines the process models and product models into a single process model and product model, respectively. If there exist concepts in the product/process models which have the same semantics but different structures, or if one concept is a specialization of another concept, then the *Refine Product/Process Models* stage is performed on the models. The need for improving and refining the current integrated model is supported by the *Remove Elements* and *Add Elements* tasks.

***Paradigm Workflow*:** The objective of this workflow is to adapt the product model produced from the paradigm model in the previous stage before creating the corresponding process model in the next stage. To achieve this goal, this workflow produces refined versions of the product models that are received as input.

***Configuration Workflow*:** If configuration characteristics are defined or modified in the previous stage, it is necessary to *Define/Modify the Configuration Package* in order to satisfy them. We must also be able to handle the lack of Configuration

Templates or Configuration Packages; the *Define/Modify Configuration Template* and *Define/Modify Configuration Package* stages are provided to cope with this problem.

Develop Method Stage. After selecting and configuring the appropriate construction blocks, the next step is to develop the method. This stage uses the input blocks to either produce a methodology or a new composite block (Fig. 5(b)). For instance, the assembly workflow receives method chunks and assembles them into a new method chunk. The *Manage Non-Functional Requirements* stage is executed in all workflows of this stage to ensure the satisfaction of non-functional requirements.

***Assembly Workflow*:** After producing the method chunks, the next step is to assemble them. This process aims at connecting and ordering the method chunks. The *Adapt the Process* task unifies the names of similar process elements and performs the transformations required to create the adapted process models. Similarly, *Adapt the Product Model* performs a similar operation on the product models. After creating the adapted process/product models and removing the inconsistencies, the *Connect Process/Product Models* task determines the order in which the process chunks must be executed, and provides the links between them. After connecting the chunks, the *Remove Elements* and *Add Elements* tasks refine and perfect the developed increment.

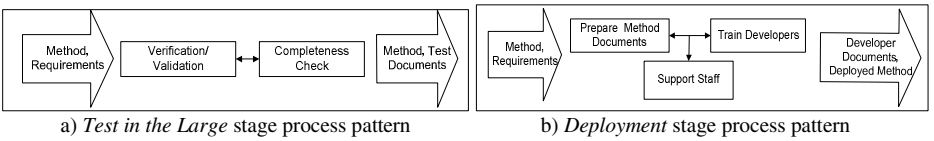
***Paradigm Workflow*:** The objective of this workflow is to create the process models corresponding to the produced product models. A process model can take various forms, i.e. activity-oriented, pattern-based, context-driven, and strategy-based. Despite the different forms a process my take, the tasks required for creating process models are the same. The creation process forms the types so that they only differ as to the details of the tasks. Therefore, we will refer to all of these forms as process elements. Identifying the process elements corresponding to products can be done through the *Situation-based Identification* and *Goal-based Identification* tasks. In the former, a typical situation method is considered as the policy for finding the process elements, while in the latter, the generic goal in the context of the method is considered as the policy. After finding the process elements, refinement is performed on the process elements identified through the *Refine Process/Product Models* stage. The *Precedence* task examines the set of process elements to find the precedence dependencies among them. After identifying the set of process elements, the main process model of the methodology must be created through connecting these process elements according to the product models. We may also have to address seamlessness, typically through creating new process elements to fill the seams. This activity is performed during the *Product-driven Process Element Definition* and *Goal-driven Process Elements Definition* tasks.

***Configuration Workflow*:** Once the base method is found, and the configuration packages/templates needed are retrieved or created, we apply the packages/templates to the base method in order to create the target methodology. Minor adjustments are then applied. If major/recurring adjustments are required, this might be an indication that a configuration template is missing, and hence, we do not have a sufficiently good match between the project situation and the existing configuration templates.

Generalization Stage. The aim of this stage is to make the produced construction blocks reusable (Fig. 5(c)). Reusable blocks are identified, made reusable through abstraction and generalization, and documented and stored for future use.

Test in the Large Stage. The objective of this stage is to perform the final tests on the method in order to find inconsistencies and deficiencies. The requirements and the generated methodology are received as input (Fig. 6(a)). The scope of evaluation is the whole methodology. We may need to send back the defects found to the previous phase to be fixed. The *Verification/Validation* task is first executed to check whether the generated methodology is free from defects and inconsistencies and whether it is in compliance with the requirements established in the requirements stage. The *Completeness* task checks whether the produced method is complete. This stage produces the tested method and test documents (including the test suite) as output.

Deployment Stage. During this stage, the project-specific method is introduced into the user environment. It receives the method produced and the requirements, and produces method documents and the deployed method (Fig. 6(b)). *Prepare Method Documents, Train Developers,* and *Support Staff* are the main tasks of this stage.



a) *Test in the Large* stage process pattern b) *Deployment* stage process pattern

Fig. 6. Components of *Deployment* phase process pattern

Refine Product/Process Models Stage. The inputs of this stage are the product/process models, requirements, and method architecture (Fig. 7). The refined product/ process models are delivered as output. Sometimes, there are concepts in the product/process models which have the same semantics but different structures; the *Product/Process Generalization* task generalizes such concepts into a new one. The *Product/Process Specialization* task is performed when one concept represents a specialization of another. The *Product/Process Aggregation* task strives to aggregate product/process elements based on the assumption that the aggregate element might satisfy the requirements not addressed by the individual constituents. The *Product/Process Decomposition* task selects the relevant product/process sub-elements and eliminates the inadequate/unwanted ones. The *Product/Process Linking* task connects the different elements through applying generalization/specialization.

Define/Adapt the Configuration Template Stage. The objective of this stage is to define a configuration template as an aggregate of configuration packages (Fig. 8). It

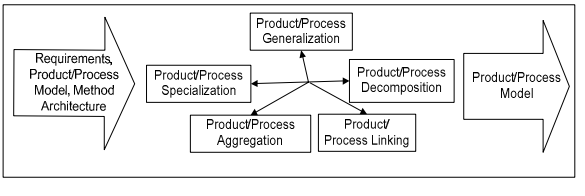


Fig. 7. *Refine Product/Process Models* stage process pattern

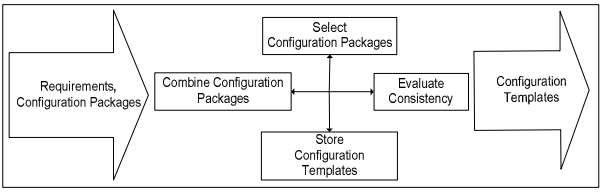


Fig. 8. *Define/Adapt the Configuration Template* stage process pattern

receives configuration characteristics and produces configuration templates. The *Selecting Configuration Packages* task elicits relevant configuration packages for a configuration template based on a development situation's characteristics. The *Combining Configuration Packages* task combines the configuration packages and resolves their classification conflicts. The configuration template's *Consistency* is then evaluated, with the result stored in the repository of configuration templates.

Manage Non-Functional Requirements Stage. This stage aims at applying the non-functional requirements to the method under construction (Fig. 9). We use the approach proposed in [16], which uses a concept called *Method Tactic*: A technique for method engineering aimed at achieving specific method qualities. For each non-functional requirement, the factors which affect it are identified through the *Analyze Non-Functional Requirements* task. For instance, information flow efficiency (speed, responsiveness and leanness) and task interdependency are factors that affect the agility and scalability requirements. At the end of this task, we attain a set of factors that can affect the non-functional requirements. The preliminary list of method tactics is identified through inventing techniques for manipulating these factors to obtain the desired effect. As expected, most tactics affect different method qualities in conflicting directions. Tactics are ultimately applied to the method or part of it in order to satisfy the non-functional requirements.

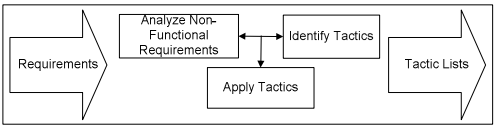


Fig. 9. *Manage Non-Functional Requirements* stage process pattern

Table 1. Essential requirements of a real-time methodology [17, 18]

Type	Requirements	No.
General Requirements	Specification at high abstraction level, possibly in a single environment	1
	General activities of software development (requirements engineering, analysis, design, implementation, test, deployment)	2
	Reuse of earlier designs	3
	Traceability to requirements	4
	Manage and monitor methodology	5
Situational Requirements	Formal Verification (rather than test) at each abstraction level	6
	(Semi-) Automatic refinement between abstraction levels	7
	Concurrency	8
	Clear separation of concerns	9
	Configuration of software and hardware	10

6 Example: Engineering a Real-Time SDM

In this section, we demonstrate the enactment of SMEP through an example. In this example, the objective is to develop a general methodology for real-time systems development. This domain was chosen because of its relative maturity, thus yielding

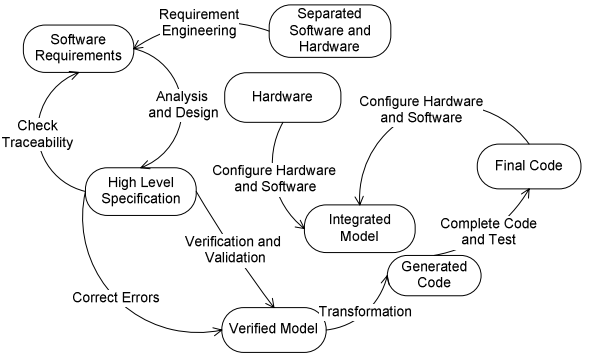


Fig. 10. Requirements model for a real-time methodology

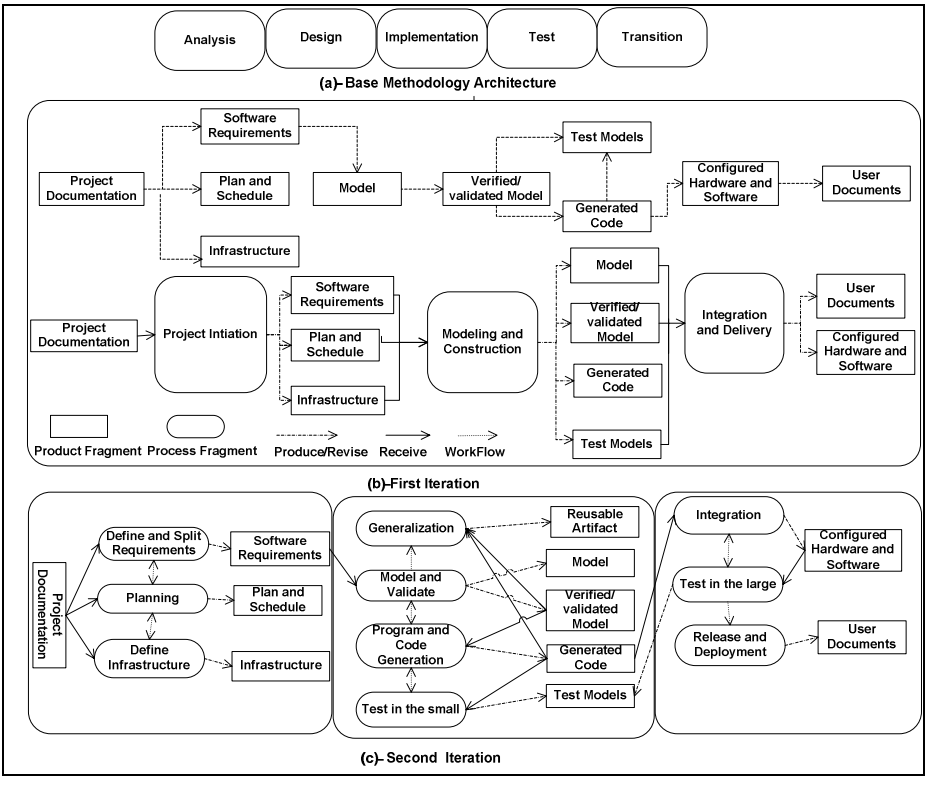


Fig. 11. Real-time methodology produced through enacting the SMEP

Table 2. Results of the third iteration: configuration of the produced methodology

Process Fragment	Sub-process Fragments	Keep?	Requirements Targeted
Define and Split Requirements	Model the Problem Domain	√	1
	Prioritize Requirements	√	1, 5
	Software Requirement Engineering	√	All except 6
	Validate Intangible Artifacts	√	3, 6
	Split Requirements	√	1, 9, 10
	System Requirement Engineering	√	All except 6
Planning	Create Project Plan	√	2, 5
	Quality Assurance Plan	√	5
	Initial Risk Assessment	√	2
	Define Initial Management Documents	√	2, 5
Define Infrastructure	Select Platform	√	3, 7
	Select Tool	√	7
	Define team	×	-
Model and Validate	Domain Analysis and Object Identification	√	2
	Object-Oriented Analysis and Design	√	2, 3, 4, 7, 8
	Validate Intangible Artifact	√	3, 6, 7
	Concurrency Modeling	√	4, 7, 8
	Architectural Design	√	3, 4, 7, 8, 9, 10
Test in the Large	Develop Plan Test	√	2
	Use Tool	×	-
	System Test	√	2
	User Acceptance Test	√	2
	Acceptance Test		2, 9
Program and Code Generation	Select Requirements for Iteration	√	4
	Understand the Model	√	4, 10
	Use Tool	√	9, 10
	Instantiate and map to physical Hardware	√	7
	Generate Code from Models	√	4, 7
	Refine Models	√	2
Test in the Small	Develop Test Plan	√	2
	Develop Test Case	√	2
	Execute Test	√	2
	Document and Analyze Results	√	2
Generalization	Recognize Reusable Artifacts	√	3
	Generalize Artifacts	√	3, 4
	Document Reusable Artifact	√	3
	Use of Pattern	√	3
Release and Deployment	Transfer Knowledge Deploy System	×	-
	Train User	√	2
	Assemble Hardware	√	9, 10
	Prepare User Documentation	√	2, 4, 11,
	Change UI Them	×	-
Integration	Verify Compatibility	√	2
	Integrated Related Parts of Subsystem	√	2
	Create Software Package	√	2
	Integrate Hardware Wrapper	√	9, 10
	Map software to physical Hardware	√	9, 10
	Validate hardware Specifications	√	9, 10
	Check Hardware Capability	√	9, 10

better-defined requirements. Since real-time software tends to be closely intertwined with hardware elements, the constraints involved are profoundly different from those typically seen in data-intensive information systems. We will use an unabridged (uncustomized) version of SMEP, enacted phase by phase in order to show the efficacy of the process. The deployment phase, however, is beyond the context of this example.

Method initiation phase: As mentioned earlier, this phase produces the methodology requirements and architecture. The requirements expected to be satisfied by a real time systems development methodology are shown in table 1 [17, 18], with the corresponding requirements model illustrated in Fig. 10. As shown in Fig. 11(a), the generic software development process [1] is taken as the base methodology.

Method construction phase: This phase aims at creating a methodology which satisfies the requirements model as well as any non-functional requirements. It can be achieved by performing one or more SMEP workflows. Firstly, the paradigm workflow is selected according to the requirements. This workflow selects one of the metamodels available – including OPF [19] or SPEM-2 [20] – or generic object oriented models such as OOSP[8] and RTSP [17], and then instantiates it to produce the product model and its corresponding process model. The process model is created in the form of a pattern-based process in which patterns are at a high level of granularity.

The result of this iteration is shown in Fig. 11(b). The second iteration uses the assembly workflow to enrich the pattern-based process created in the previous iteration. Based on the requirements and the patterns used, process fragments are retrieved from the method base and connected to each other to realize the context of their relevant patterns. Fig. 11(c) shows the result of the second iteration. Each process fragment selected in this iteration consists of sub-process fragments at finer levels of granularity. In the third iteration, the configuration workflow is applied to each process fragment, pruning and fine-tuning the process so that the requirements are fully satisfied. Table 2 shows which parts of each process fragment are kept and which parts are deleted as a result of the third iteration.

7 Conclusions

We have proposed a pattern-based framework for Situational Method Engineering. Also proposed are a set of method-engineering-specific process patterns that can be used for constructing method engineering approaches. Pattern extraction was based on the detailed inspection of prominent method engineering approaches. The patterns have been organized into an instantiated version of the proposed pattern-based SME framework, thus yielding a generic Situational Method Engineering Process (SMEP). This work can be further extended to investigate the full details of the task process patterns. Future work can then be directed towards developing a Computer-Aided Method Engineering (CAME) environment [2], to be used during methodology construction/adaptation for pattern-based methodology development.

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. Ralyté, J., Deneckére, R., Rolland, C.: Towards a generic model for situational method engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 95–110. Springer, Heidelberg (2003)
3. Ramsin, R.: The Engineering of an Object-Oriented Software Engineering Methodology. Ph.D. Thesis, University of York, York, UK (2006), http://www.cs.york.ac.uk/ftpdir/reports/YCST-2006-12.pdf
4. Ralyte, J., Rolland, C., Ayed, M.B.: An Approach for Evolution Driven Method Engineering. In: Krogstie, J., Halpin, T., Siau, K. (eds.) Information Modeling Methods and Methodologies. Idea Group Inc, USA (2003)
5. Karlsson, F., Gerfalk, P.J.: Method configuration: adapting to situational characteristics while creating reusable assets. Information and Software Technology 46(9), 619–633 (2004)
6. Ralyte, J., Rolland, C.: An assembly process model for method engineering. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 267–283. Springer, Heidelberg (2001)
7. Paige, R.F., Brook, P.J.: Formal Agile Method Engineering. In: Romijn, J.M.T., Smith, G.P., van de Pol, J. (eds.) IFM 2005. LNCS, vol. 3771, pp. 109–128. Springer, Heidelberg (2005)

8. Ambler., S.W.: Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press, Cambridge (1998)
9. Gnatz, M., Marschall, F., Popp, G., Rausch, A., Schwerin, W.: Modular Process Patterns supporting an Evolutionary Software Development Process. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2001. LNCS, vol. 2188, p. 326. Springer, Heidelberg (2001)
10. Tasharofi, S., Ramsin, R.: Process Patterns for Agile Methodologies. In: Advanced Programming Environments. IFIP, vol. 244. Springer, Heidelberg (2007)
11. Prakash, N., Bhatia., M.S.P.: Generic models for engineering methods of diverse domains. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 612–625. Springer, Heidelberg (2002)
12. Brinkkemper, S.: Formalisation of information systems modeling. Ph.D. thesis, University of Nijmegen. Thesis Publishers, Amsterdam (1990)
13. Osterweil., L.J.: Software processes are software too. In: 9th international Conference on Software Engineering, pp. 2–13. IEEE Computer Society Press, Los Alamitos (1987)
14. Coulin, C., Zowghi, D., Sahraoui, A.-E.-K.: A Lightweight Workshop- Centric Situational Approach for the Early Stages of Requirements Elicitation in Software Systems Developme. In: Proceedings of the International Workshop on Situational Requirements Engineering Processes (SREP 2005), France (2005)
15. Ralyte, J.: Requirements definition for the situational method engineering. In: Proceedings of the IFIP WG8.1 working conference on engineering information systems in the internet context (EISIC 2002), pp. 127–152 (2002)
16. Zhu, L., Staples, M.: Situational Method Quality. In: IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME 2007) LNCS, vol. 244, pp. 193–206. Springer, Heidelberg (2007)
17. Esfahani, N.: Introducing a set of process patterns for real-time software, MSc Thesis (in Persian), Department of Computer Engineering, Sharif University of Technology, Tehran, Iran (2008)
18. Cuccuru, A., De Simone, R., Saunier, T., Siegel, G., Sorel, Y.: P2I: An Innovative MDA Methodology for Embedded Real-Time System. In: Proceedings of the 8th Euromicro Conference on Digital System Design, pp. 26–33 (2005)
19. Firesmith, D., Henderson-Sellers, B.: The OPEN Process Framework: An Introduction. Addison-Wesley, Reading (2001)
20. Object Management Group, Systems and Software Process Engineering Metamodel v2.0 (SPEM), OMG (2007)