Requirements Eng (2006) 11: 58–78
DOI 10.1007/s00766-005-0019-0

ORIGINAL ARTICLE

Isabelle Mirbel · Jolita Ralyté

Situational method engineering: combining assembly-based and roadmap-driven approaches

Received: 14 July 2004 / Accepted: 9 September 2005 / Published online: 3 November 2005

Abstract Because the engineering situation of each information system development (ISD) project is different, engineering methods need to be adapted, transformed or enhanced to satisfy the specific project situation. Contributions, in the field of situational method engineering (SME), aim at providing techniques and tools allowing to construct project-specific methods instead of looking for universally applicable ones. In addition to the engineering method tailoring, necessary to fit the project situation, a customization of the engineering method for each engineer participating in the project is also required. Such a configuration allows a better understanding of the method by focusing on guidelines related to the project engineer's daily tasks. It also increases his/her involvement in the ISD method realization. To achieve this twofold objective (ISD method tailoring and customization), we propose a framework for SME combining an assembly-based approach for project-specific method construction and a roadmap-driven approach for engineer-specific method configuration. The first step of our process provides support to build a new method that is most suitable for the current ISD project situation, whereas the second step aims at choosing the most adapted path (roadmap) to satisfy the requirements of a particular project engineer within the project-specific method. The two core elements of our SME framework are the method chunks repository and the reuse frame. The former concerns reusable method components definition and storage whereas the latter deals with the characterization of the project situation and the project engineer's profile. In this paper we start first by presenting our SME framework and its core elements: the method chunk repository and the reuse frame. Then we show how to take advantage of them through our two-step process combining assembly-based method construction and roadmap-driven method configuration.

I. Mirbel (✉)
Laboratoire I3S, Les Algorithmes, Route des Lucioles, BP 121, 06903 Sophia Antipolis Cedex, France
E-mail: isabelle.mirbel@unice.fr

J. Ralyté
Centre Universitaire d'Informatique, Université de Genève, 24 rue du Général Dufour, CH-1211 Genève 4, Switzerland
E-mail: ralyte@cui.unige.ch

1 Introduction

Objectives, problems and requirements as well as the whole engineering environment vary from one information systems development (ISD) project to another. Besides, each project is governed by a number of organizational, technical and human factors as complexity, specificity and novelty of the application domain, degree of innovation, way of working, team expertise, etc. In other words, the engineering situation of each ISD project is different and requires project-specific methods and tools for supporting it. Even though traditional ISD methods claim to be universal and propose a large number of models and views for system analysis and specification, they cannot foresee all possible ISD situations. Moreover, many studies of the use of system development methodologies in practice have shown that the 'cookbook' approach for system engineering is not working as expected [7, 12, 13]. Methods are almost never suited literally and even more there is a wide difference between the formalized sequences of steps and stages prescribed by a method and their real application in practice [7]. This tension between the 'method-in-concept' (the method as formalized in manual) and the 'method-in-action' (as interpreted by the developer) is highlighted in [7, 19]. Although there is a common consensus that ISD methods need to be adapted, transformed, or enhanced to satisfy the development situation they have to face [3, 7, 10, 16, 33, 37, 46], there is a great difficulty to change them because of their rigidity and lack of modularity.

To resolve this problem, the discipline of *Situational Method Engineering* (SME) focuses on the creation of new techniques and tools allowing to construct project-specific methods 'on the fly' [18] instead of looking for universally applicable ones. Most of the SME approaches promote the construction and adaptation of new methods by assembling reusable *method fragments* [3] or *method chunks* [33, 34], stored in some method repository [3, 11, 28, 38, 47]. These approaches lead to the construction of new modular methods. A modular method means a collection of interconnected method fragments/chunks, which are presented and illustrated in Sect. 2 of this paper. Thanks to this modularity, methods are more flexible and adaptable. They can be modified and improved by adding new fragments/chunks or by removing the unnecessary ones in order to meet the requirements of a given situation [11, 48].

Another, a somewhat different kind of approach, called *method configuration* [16] or *process tailoring* [46], claims that many organisations choose one commercially available system development methodology and focuses on how to adapt this methodology to situational factors of the project at hand. These approaches introduce the notion of *method rationale* as a means to support method evolution. However, they also have to face the problem of method adaptability and flexibility.

Empirical studies show that method (process) tailoring is difficult in that it involves intensive knowledge generation and deployment [46]. Indeed, even if methods are decomposed into fragments through SME, ISD crew members must apprehend the method as a whole and understand all its concepts in order to use it, which can have some negative impact and discourage the ISD crew member from using methods [2]. Therefore, in addition to the tailoring of project-specific ISD methods, there is also a need for the customization dedicated to ISD project crew members [22, 25, 26]. Finally, existing approaches emphasize the development process too much compared to the organizational and human dimensions [27]. Indeed, each ISD crew member has to perform specific tasks throughout the project at hand and thus needs a specific view of the method used for this project realization.

Therefore, in this paper it is considered that each ISD project should start with the definition of its proper method that best fits its situation. Then, the obtained method can be personalized for each project crew member according to his/her tasks; only the method chunks supporting these tasks should be proposed to him/her.

Method construction starting from existing method chunks as well as method configuration to suit project crew member's situation may be seen as a specific technique to select and retrieve method components. Existing approaches supporting the search and retrieval of software components can be classified into four types [17, 21]: (1) simple keywords and string search, (2) faceted classification and retrieval, (3) signature matching and (4) behavioural matching.

Simple keyword searching may result in too many or too few items retrieved or even unrelated items. Drawback of the faceted classification search approaches is the difficulty in managing the classification scheme when domain knowledge evolves. Signature matching techniques are dedicated to software components embedding code and are difficult to apply on components providing knowledge about requirements and way of working, as it is in this case. Behavioural matching techniques are difficult to use when components have complex behaviours or involve side-effects. Moreover, it is difficult to express the required behaviours. Finally, all these techniques do not provide ways to augment or extend query [50].

Therefore, a need was identified to combine user intention and application domain information. Recent approaches [31, 50] propose means to handle and take into account knowledge about the application domain through the search and retrieval process. In this approach, the focus is on the methodology rather than the application to develop. A specific frame, is proposed which may be seen as ontology, allowing to define the meaningful knowledge related to method engineering activity and to build a method dedicated to the needs of a specific company or a specific project. This polymorphic structure also allows the method engineer to broadcast methodological ways of working otherwise than through deliverables. On the other hand, it allows method users to retrieve method components dedicated to their specific work inside the project-specific method without being obliged to understand the whole methodology.

To sum up, the objective of the work is to propose an approach for SME including both project-specific ISD method construction and its customization for each engineer participating in the project. To achieve this objective, two complementary approaches are combined (1) the *assembly-based* approach for project-specific method construction and (2) the *roadmap-driven* approach for method configuration in order to satisfy specific needs of each ISD crew member. Both approaches share the same method chunks repository and the common ISD project development knowledge frame, namely the reuse frame.

It is clear that projects executed with situation-specific methods may become incomparable for project managers who want to predict the cost of ISD projects. The global framework proposal is a means to reduce the potential divergence between different project executions by providing a common frame for all the projects in the company and by postponing the customization to the ISD crew member level, that is to say where freedom and tailoring is still needed for acceptance purposes.

The remainder of this paper is as follows: in Sect. 2 a framework for SME combining the assembly-based and roadmap-driven approaches is proposed. The elements as method chunk and reuse frame shared by the two approaches are defined and illustrated in this section. Section 3 details and illustrates the assembly-based approach for project-specific method construction as the first step of the global SME approach, while Sect. 4 deals with the roadmap-driven approach for method configuration as its second step. The conclusions and discussions about the future preoccupations are proposed in Sect. 5.

2 Framework for SME

In this section the framework for SME is presented combining two complementary method engineering approaches called *assembly-based* and *roadmap-driven*. As shown in Fig. 1 the two approaches represent two core steps in the situation-specific method construction–configuration process:

1. The aim of the first step is to build a new method that is most adapted/suitable for the current ISD project situation. This step is found on the assembly-based method engineering approach detailed in Sect. 3.
2. The second step aims at choosing the most adapted/suitable roadmap within a selected method with regards to the project and the ISD crew member's needs. The roadmap-driven method configuration approach establishes the basis of this step. This approach is detailed in Sect. 4 of this paper.

As shown in Fig. 1, both steps share two common elements: the *method chunks repository* and the *Reuse Frame*. The former concerns the reusable method components definition and storage whereas the latter deals with the characterization of the project situation and the definition of the requirements for a specific method or a roadmap in a method. The method chunks repository contains the effectively reusable method knowledge defined in terms of method chunks, while the reuse frame contains the knowledge about the reuse context of these method chunks and provides criteria for project and project engineer situation characterization.

Two core actors are identified in the framework (Fig. 1): the *method engineer* and the *ISD crew member*. Each of them has a specific role in the SME process. The method engineer is in charge of executing step 1 while the ISD crew member is responsible for the realization of the step 2. In other words, the method engineer deals with the project specific method construction whereas the ISD crew member selects in this method the roadmap corresponding to his or her tasks in the ISD project realization.

Besides, the method engineer is responsible for filling up the method chunks repository, and the definition and management of reuse criteria in the reuse frame.

2.1 Method chunks repository

One of the core elements in the SME framework is the repository of method chunks (Fig. 1). The role of this repository is to store reusable parts of methods as prospective method building blocks.

Based on the observation that any method has two interrelated aspects, product and process, several authors propose two types of method components: process fragments and product fragments [3, 11, 32]. Other authors consider only process aspects and provide *process components* [6, 8], or *process fragments* [22, 26]. In this approach these two aspects are integrated in the same module that is called *method chunk* [28, 33, 34, 42].

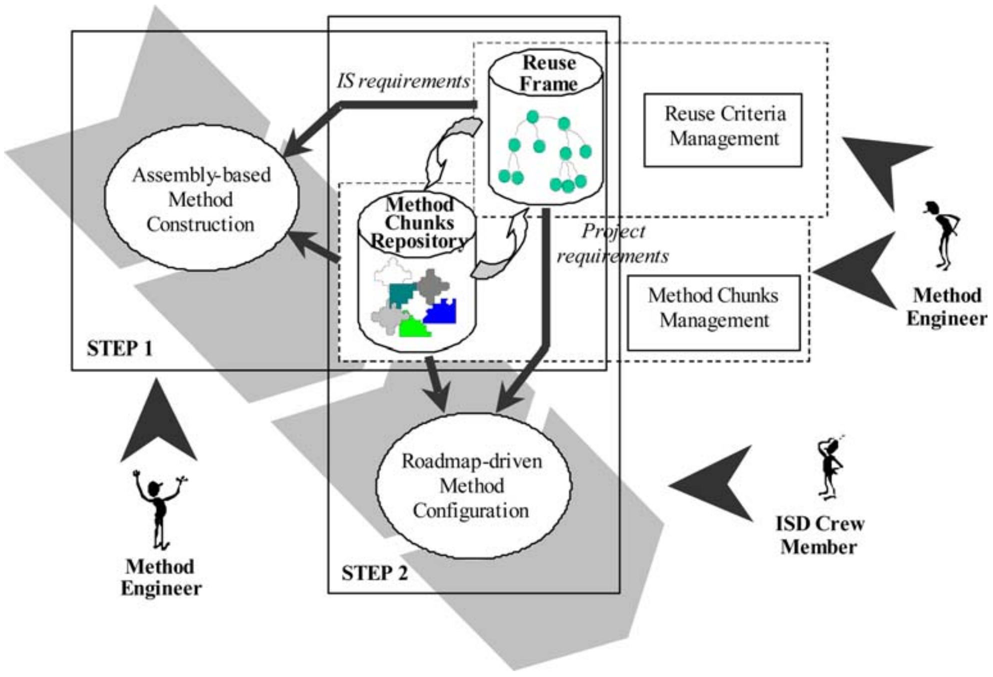


Fig. 1 A framework for situational method engineering combining assembly-based and roadmap-driven approaches

The notion of *method bloc* proposed by Prakash [29] is similar to the method chunk as it also combines product and process perspectives into the same modelling component. Both of these notions, *method fragment* and *method chunk*, represent the basic blocks for constructing 'on the fly' methods. Generally, they are stored in some method repository or method base [10, 38, 42, 47].

Van Slooten and Brinkkemper [48] combine method fragments into *route maps*. A complete route map represents a system development method. In this approach, the step 2 dealing with roadmap-driven method configuration uses the notion of the route map to represent the specific path dedicated to a particular ISD crew member within a selected method [26], that is to say a subset of an already combined set of method chunks.

Another kind of SME approach uses generic *conceptual patterns* for method construction and extension [39, 40], which capture generic lows governing the construction of different but similar methods. *Decision-making patterns* capturing the best practices in enterprise modelling are proposed by Rolland [41] to support enterprise knowledge development process. Deneckere and Souveyet [5] propose *domain-specific* process and product *patterns* for existing method extension.

In line with all these propositions around the notion of the reusable method component, they are classified into *method chunks*, *method fragments*, *patterns* and *roadmaps* (Fig. 2).

In this approach two types of method components are used: method chunks and roadmaps. A method chunk represents a reusable building block for situation-driven method construction or adaptation whereas a roadmap represents a path in a method or a specific sequence of method chunks in a method.

2.1.1 *Method chunk*

A method is considered as a couple of two interrelated models: *product model* and *process model*. The product model of a method defines a set of concepts, relationships between these concepts and constraints for a corresponding schema construction. The process model describes how to construct the corresponding product model. Moreover, a method in this approach is viewed as a set of loosely coupled method chunks expressed at different levels of granularity. A method chunk is an autonomous and coherent part of a method supporting the realization of some specific ISD activities. Such a modular view of methods favours their adaptation and extension. Moreover, this view permits to reuse chunks of a given method in the construction of new ones.

As a part of this method, a method chunk ensures a tight coupling of some process part of a method process model and its related product part. In the product part, also called product fragment, the product to be delivered by the method chunk is captured whereas in the process part, also called process fragment, the guidelines allowing to produce the product are given. For example, the method chunk providing guidelines (process part) for the use case model [14] construction should also provide definitions of the concepts as actor, use case, scenario, extend relationship, etc. (product part) used in this model. As shown in Fig. 3 which represents the notion of the method chunk, the corresponding product and process parts constitute the *body* of the method chunk.

The *interface* of the method chunk captures the reuse context in which the method chunk can be applied. It is formalized by a couple <*situation, intention*>, which characterizes the situation that is the input of the chunk process and the intention (the goal) that the chunk achieves. "To construct a use case model" is an example

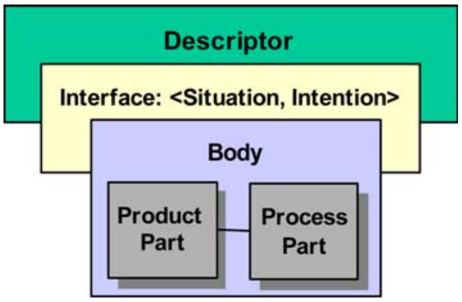
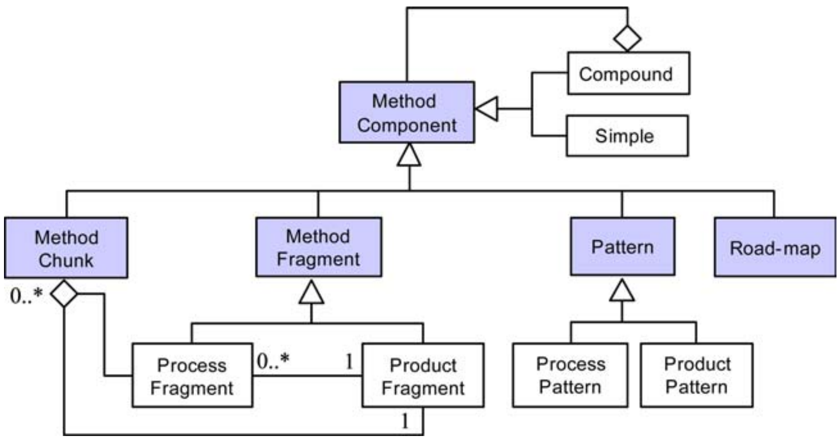


Fig. 3 Notion of the method chunk

Fig. 2 Typology of reusable method components



of the intention associated to the method chunk providing guidelines for the use case model construction; the "problem statement" specifies the situation in which this chunk can be applied that is the input product necessary to start the execution of this method chunk.

Besides, a *descriptor* is associated to every method chunk. It extends the contextual view captured in the chunk interface to define the context in which the chunk can be reused. The descriptor takes values from the reuse frame and associates them to the corresponding method chunk.

Figure 4 shows the meta-model for modular methods. According to this meta-model, a method is also viewed as a method chunk of the highest level of granularity. The body of the method chunk captures a part of method process model called *guideline* that can be considered as autonomous and reusable and a part of its product model needed to perform the process encapsulated in this guideline.

A guideline is defined as 'a statement or other indication of policy or procedure by which to determine a course of action' [1]. For us, a guideline embodies *method knowledge* to guide the application engineer in achieving an intention in a given situation. Therefore, the guideline has an *interface*, which describes the conditions of its applicability (the situation) and a *body* providing guidance to achieve the intention, that is, to proceed in the construction of the target product. As mentioned previously, the interface is a couple <*situation, intention*>. A situation represents an input to the method chunk process and contains one or several *product elements* necessary to start its execution in order to achieve its intention. An intention represents an engineering goal to be achieved by applying the method chunk. An intention is expressed following a linguistic approach proposed by Prat [30] as a clause with a *verb* and a *target*. It can also have several *parameters*, where each parameter plays a different role with respect to the verb. For example, in the intention "Construct a use case model following OOSE method advices", the verb "Construct" is followed by the target product "a use case model" and the manner to achieve this intention "following OOSE method advices". It is formalized as follows: "Construct$_{\text{Verb}}$ (a use case model)$_{\text{Target}}$ (following OOSE method advices)$_{\text{Mannner}}$".

The product elements used in the method chunk situations as well as the verbs and target products specified in the method chunks intentions are defined in the method engineering glossary. They are used in addition to the reuse context during the retrieval process of the method chunks from the repository.

The body of the guideline details how to apply the chunk to achieve its intention. The interface of the guideline is also the interface of the corresponding method chunk. Guidelines in different methods have different contents, formality, granularity, etc. In order to capture this variety, three types of guidelines: simple, tactical and strategic are identified.

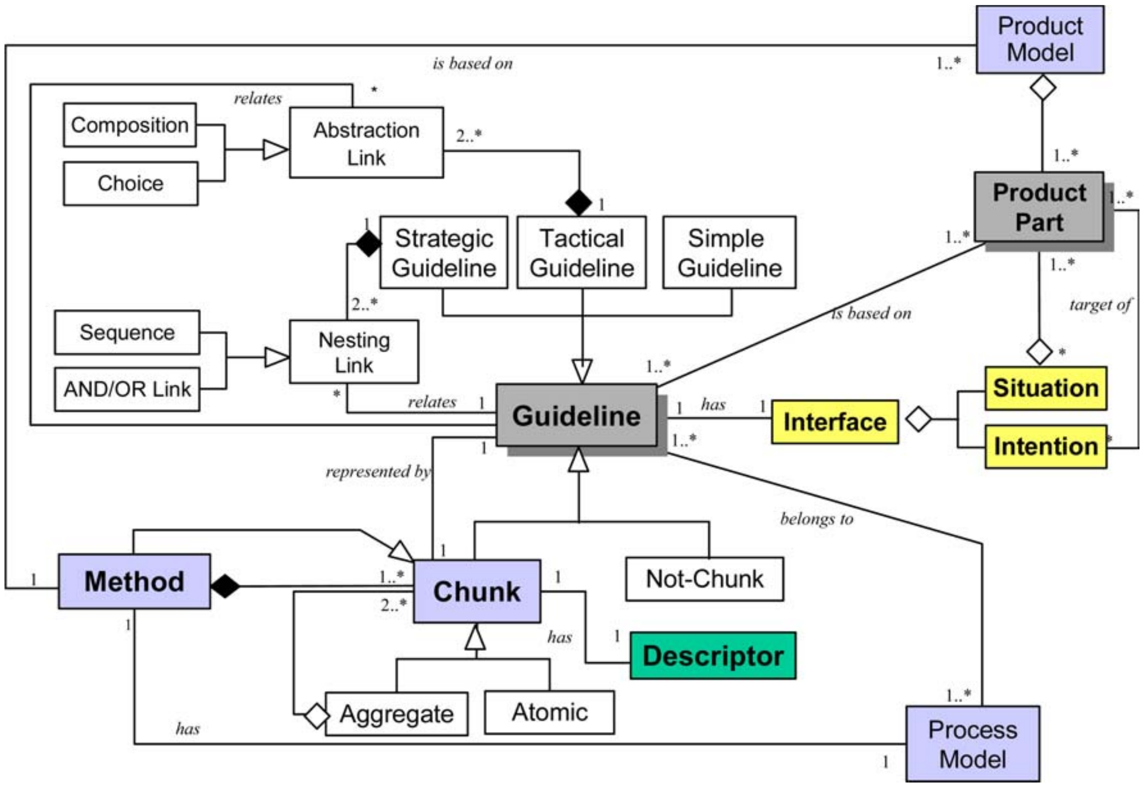


Fig. 4 The meta-model for modular methods

A simple guideline may have an informal and textual content advising on how to proceed to handle the situation in a narrative form. It can also be a simple *executable* action leading to some transformation of the product under construction. Figure 5 illustrates a method chunk with a simple informal guideline proposed by the *L'Ecritoire* method [44]. The role of this method chunk is to help the requirements engineer in scenario writing process by providing him with style guidelines. The product part of this chunk defines the "Scenario" concept and its structure.

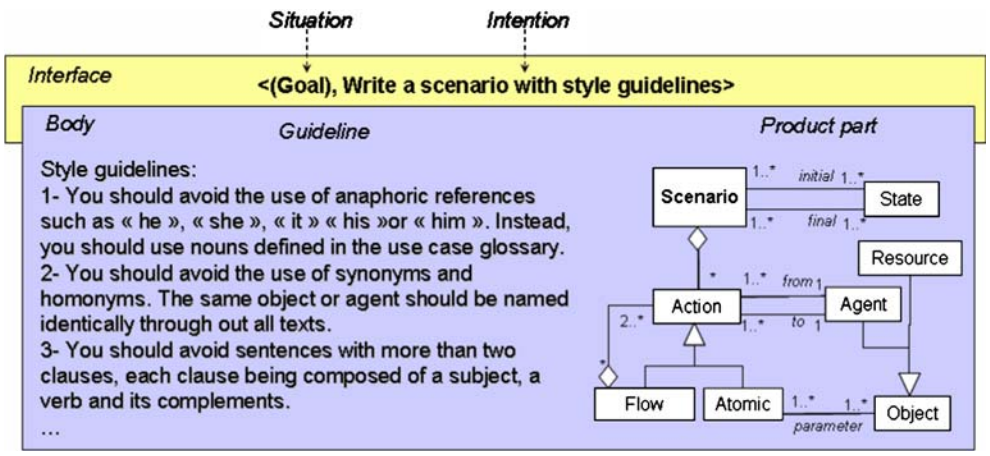
A *tactical guideline* is a complex guideline composed of a set of steps. The *NATURE* process modelling formalism [15] is used to define tactical guidelines. This formalism is based on a tree structure. A high level guideline is decomposed into a set of more detailed guidelines, which can also be decomposed until executable actions are obtained. There are two possible types of decomposition: the *choice* (a possibility to select one alternative among several possibilities) and the *plan* of steps to realize. Figure 6 illustrates a method chunk with a tactical guideline for a use case model construction. This guideline is defined as a plan composed of four sub-guidelines: "<(Problem Statement), Identify an actor>*, <(Problem Statement, Actor), Identify a use case>*, <(Problem Statement, Use case), Describe use case> and <(Use case model), Refine use case model>*". Moreover, each of these sub-guidelines is defined as a plan or a choice of several sub-guidelines. The 'star' mark next to the guideline definition means that the corresponding guideline can be repeated *N* times.

A s*trategic guideline* is a complex guideline called a *map* [45] that uses a graph structure to relate its sub-guidelines. Each sub-guideline belongs to one of the three types of guidelines. A s*trategic guideline* provides a strategic view of the development process telling which *intention* can be achieved following which *strategy*. An *Intention* is a goal that can be achieved by the performance of an activity (automated/semi-automated or manual). There are two special intentions *start* and *stop* that allow to begin and end the progression in the map, respectively. An intention is expressed following a linguistic approach mentioned previously [30]. A *strategy* is an approach, a manner to achieve an intention. Thus, a map is a label directed graph in which the nodes are the intentions and the edges between the intentions are the strategies.

The map permits to represent a process allowing several different ways to develop the product. A set of guidelines is associated to the map. They help the application engineer to progress in the map and to achieve the intentions following selected strategies. More exactly, a map is a composition of a set of sections where a section is a triplet <*Source intention, Target intention, Strategy*>. Every section provides an *intention achievement guideline* (IAG) indicating how to achieve the target intention following the strategy given the source intention has been achieved. Two other types of guidelines, *intention selection guideline* (ISG) and *strategy selection guideline* (SSG), help to progress in the map that is to select the next intention and to select the next section, respectively.

To illustrate the strategic guideline it is proposed to consider the *L'Ecritoire* approach [43, 44, 51, 52]. This method chunk provides guidelines to discover functional system requirements expressed as goals and to conceptualize these requirements as scenarios describing how the system satisfies the achievement of these goals. As shown in Fig. 7, the corresponding guideline is represented by a map with three core intentions: "Elicit a goal", "write a scenario" and "conceptualize a scenario". Several different strategies are provided by the chunk to achieve these intentions each of them representing a different manner to do it. This method chunk is an aggregate one; the IAG of each of its process map sections is represented by another method chunk of the lower level of abstraction. For example, the IAG associated to the map section <Elicit a goal, Write a scenario, Free prose strategy> is in fact another method chunk that proposes a set of detailed style and content guidelines supporting scenario authoring.

As mentioned previously, each method chunk has a *descriptor,* which captures the knowledge about its reuse conditions. In other words, the knowledge contained in



Style guidelines:
1- You should avoid the use of anaphoric references such as « he », « she », « it » « his »or « him ». Instead, you should use nouns defined in the use case glossary.
2- You should avoid the use of synonyms and homonyms. The same object or agent should be named identically through out all texts.
3- You should avoid sentences with more than two clauses, each clause being composed of a subject, a verb and its complements.
...

Fig. 5 Example of a method chunk with a simple informal guideline

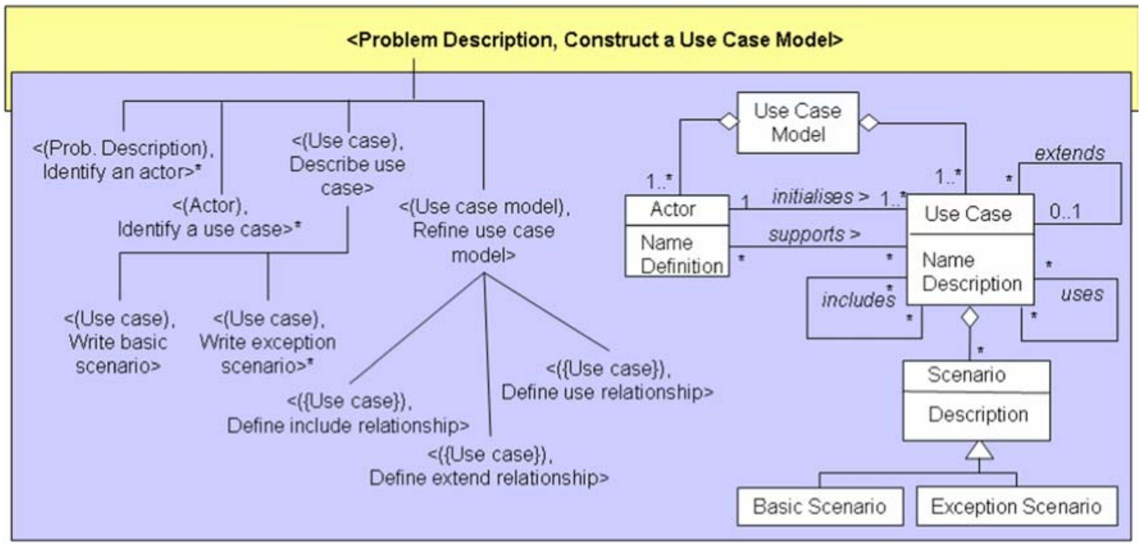

<Problem Description, Construct a Use Case Model>
<(Prob. Description), Identify an actor>*
<(Use case), Describe use case>
<(Actor), Identify a use case>*
<(Use case model), Refine use case model>
<(Use case), Write basic scenario>
<(Use case), Write exception scenario>*
<({Use case}), Define use relationship>
<({Use case}), Define include relationship>
<({Use case}), Define extend relationship>
Use Case Model
Actor
Name
Definition
initialises >
supports >
Use Case
Name
Description
extends
includes
uses
Scenario
Description
Basic Scenario
Exception Scenario

Fig. 6 Example of a method chunk with a tactical guideline

the descriptors is used during the method chunks selection and retrieval from the repository. As shown in Fig. 8, the two key elements of the descriptor are the *reuse context* and the *reuse intention*. The reuse context captures a set of criteria taken from the reuse frame and characterising the corresponding method chunk. For example, the reuse context of the method chunk presented in Fig. 7 should specify that this chunk is applicable during the requirements elicitation activity. In the next section more details are provided about the reuse frame and how it allows the enhanced retrieval of method chunks in order to better suit the method engineer's and ISD crew member's needs.

The reuse intention expresses the objective that the method chunk helps to satisfy in the corresponding engineering activity. The reuse intention has the same structure as the chunk intention that is defined by a verb, a target and a set of parameters, which are specified in the glossary. For example, the reuse intention of the method chunk presented in Fig. 7 is defined as ''Discover$_{\text{verb}}$ (functional system requirements)$_{\text{Target}}$ (following L'Ecritoire strategy)$_{\text{Manner}}$''.

Besides, the descriptor contains the *name* and the *ID* of the corresponding method chunk and a narrative description of its *objective*. It also specifies the *type* (i.e. atomic or aggregate) and identifies the *origin* of the chunk (i.e. the method from which this chunk is extracted). If the method chunk is an aggregate one, its descriptor specifies its components and vice versa, if the chunk takes a part of some aggregate chunks its

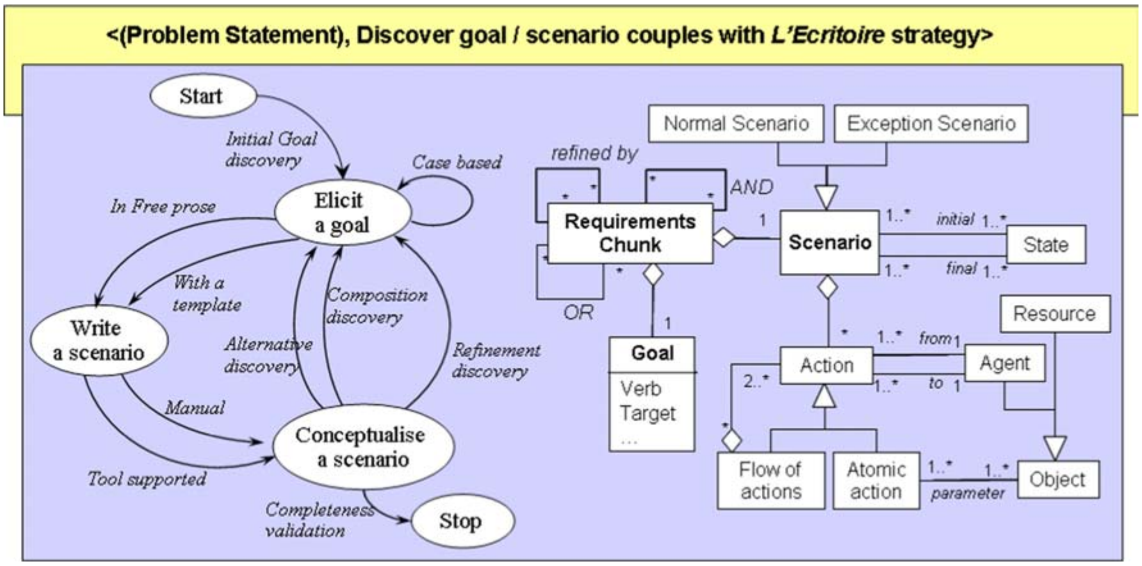


Fig. 7 Example of a method chunk with a strategic guideline

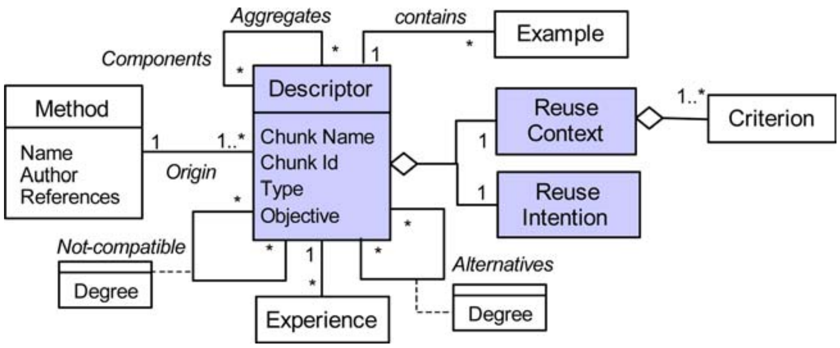


Fig. 8 The descriptor

descriptor identifies them. The *aggregates/components* relationship allows to navigate from one descriptor to another during the selection process without looking at the content of the corresponding chunks. The *alternative* and *not-compatible* method chunks to the given one can also be specified in the descriptor indicating the degree of complementarity and incompatibility respectively. Some application *examples* and *experience* reports can be added in order to facilitate method chunks selection in the case where several chunks would be retrieved to satisfy the same intention.

2.1.2 Roadmap

A roadmap is composed of one or several coherent sequence(s) of method chunks taken from a situational method described in terms of chunks and corresponding to the usage of this method by a particular ISD crew member. It is a customized view of a situational method to answer the specific need of an ISD crew member. Three categories of customized view of a situational method are described:

- The ISD crew member needs help on a specific step of the ISD method: for instance, he/she looks for the method chunks dealing with the requirement elicitation activity. In such a case, a set of organized chunks covering specific required stages of the ISD process is provided.
- The ISD crew member asks for help on a specific point of the ISD method: for instance he/she looks for the guidelines to build a class diagram by using the UML notation. In this case, a small set of chunks, not related among them, is provided.
- The ISD crew member wishes the situational method to be presented from a specific view point: for instance, he/she looks at guidelines related to the fact that the ISD takes place on top of a legacy application. In this case, a large set of chunks covering as much as possible the ISD process, with regards to the chosen view point, is provided.

Through the roadmap notion, the goal is not to provide ISD crew members with full and detailed description of all the tasks they will have to deal with during the ISD because spaces of creativity are also required, but to give them advices when they ask for it and/or when guidelines about similar situations have been accumulated in the repository. Roadmaps are used in the second step of the approach dealing with the configuration of a situational method for an ISD crew member. The purpose in this step is not to provide another way to build a method, but to propose different ways to make the situation-specific method usable and useful ‘on the fly’ for a specific ISD crew member.

2.2 Reuse frame

Looking at the way ISD methods are used in practice, it is noticed that they are always adapted: steps are added or removed or skipped and so on. Different factors related to the project, the technology, the team expertise and the business domain lead to this tailoring. Method tailoring is supported by the assembly of predefined method chunks [3, 33]. Dedicated efforts have been made, in the field of method engineering, to provide efficient classification and retrieving techniques to store and retrieve method chunks. Classification and retrieving techniques are currently based on structural relationships among chunks (specialization, composition, alternative, etc.) and reuse intention matching.

From this point of view, current classification and retrieving means do not fully exploit the potential of breaking down ISD methods into method chunks and tailoring them. Therefore, the notion of *reuse frame* is proposed that is detailed in the following sections.

2.2.1 A reuse frame to handle critical aspects of information systems

It is believed that knowledge about organizational, technical and human factors, which is critical knowledge about ISD [4], should be taken into consideration in addition to structural knowledge and reuse intention. It allows to better qualify method chunks when entering them into the repository and to enable the use of more powerful matching techniques to retrieve them when looking at similar ISD methodological problems. It also allows to better express methodological needs for a specific ISD project, improving this way the chance to get adequate and useful method chunks.

Therefore, a *reuse frame* aggregating different ISD critical aspects useful to tailor ISD methods with regards to the organizational, technical and human dimensions of information systems (IS) is proposed. This polymorphic structure allows to construct a faceted classification of reusable assets dedicated to method engineering. Of course, method chunks, as well as methodological needs, may be defined more or less precisely with regards to ISD critical aspects. Therefore, these three aspects have to be refined and presented in a way supporting the required polymorphism, which is the case of our reuse frame which is a tree of successively refined aspects. By providing such an ontological structure, the panel of means is enlarged for method engineers to broadcast ways of working which is most of the time reduced to deliverables. And ISD crew members are provided with additional information to find the most suitable information with regards to his/her ISD methodological need (or problem).

ISD critical knowledge is described in terms of *aspect*, belonging to *aspect families*, which are successive refinements of the three main factors of IS: *human*, *organizational* and *application domain*. Starting from these three basic dimensions, each company may populate the reuse frame with its own relevant criteria in order to let ISD crew members to adapt the project (or company)-specific method in a meaningful way for the company. In the global framework, a reuse frame content is also provided that was built from various works made on meaningful criteria for method customization. With regards to the *organizational* dimension, it was started from the work of van Slooten and Hodes providing elements to characterize ISD projects [49]: contingency factors, project characteristics, goals and assumptions as well as system engineering activities. With regards to the *application domain* dimension, it was started from previous work on *JECKO*, a context-driven approach to software development developed in collaboration with the Amadeus Company and including a contribution to define software critical aspects in order to get suitable documentation to support software development process [22, 23]. The application domain dimension also includes aspects related to source system (as legacy system are more and more present in companies) and application technology, which requires more and more adapted development processes. And finally, concerning the *human* dimension, means are provided to qualify the different kinds of method users who may be involved in the ISD project (analysts, developers, etc.) as well as their expertise level.

All the knowledge about ISD critical aspects is included into the reuse frame taking the form of a tree, where leafnodes are aspects and intermediary nodes are families. Indeed, nodes close to the root node are seen as general aspects while nodes close to leaf nodes (including leafnodes) are seen as precise aspects. The top of the reuse frame is shown in Fig. 9. Refer to the annex for detail of each branch.

A leafnode (or aspect) is defined through a *name*. An intermediary node (or family) is also defined through a *name* and completed by information about relationships among the different aspects or sub-families belonging to it. This additional information, aiming at better specifying the way aspects belong to a family, helps in using them to constitute a coherent context. Two kinds of information are provided:

- The *classified* field indicates if direct aspect or sub-families are classified (*cl* = *yes*) or not (*cl* = *no*). For instance, if Code Reuse is a *family*, Weak Code Reuse, Medium Code Reuse and Strong Code Reuse are aspects, which are classified (Weak Code Reuse means to reuse less code than when Medium Code Reuse is indicated, which again means less reuse than if Strong Code Reuse is chosen). It is interesting to indicate this information because when retrieving chunks

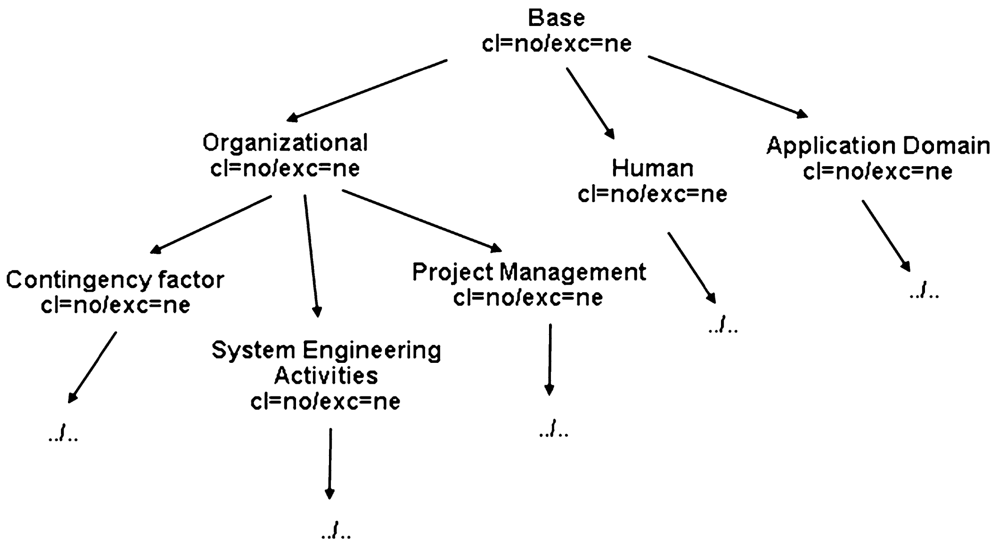


Fig. 9 Reuse frame

associated with the Medium Code Reuse aspect for instance, it may also be interesting to look at chunks associated with the Weak Code Reuse or Strong Code Reuse aspects.
- The *exclusion* field indicates whether direct aspects or sub-families are exclusive ($exc = e$) or not ($exc = ne$). For instance, there is in the reuse frame an aspect related to project time pressure. Guidelines may be given for projects under high time pressure as well as projects under low time pressure. Therefore, *time pressure* is a *family* and low time pressure and high time pressure are aspects. These are specified as exclusive aspects because guidelines associated to high time pressure projects are not compatible with guidelines associated with low time pressure projects and could not be provided in the same project-specific method or roadmap.

Indeed, the *reuse frame*, *RF*, is a tree where:

- the root node is defined as <name = base, exc = ne, cl = yes, type = root>,
- non-leaf nodes are family defined as *<name, exc, cl, type = family>*,
- leaf nodes are aspects defined as *<name, type = aspect>*

where *exc* = *e* or *ne*, respectively for exclusive or non-exclusive and *cl* = *yes* or *no* for classified or non-classified.

As critical aspects of ISD may evolve through time, means have to be provided to make the reuse frame content evolve too. Method engineers could make the reuse frame evolve by specifying more deeply existing aspects by refining them (i.e. transforming aspects into families grouping new aspects) or by reorganizing existing families by adding new intermediary nodes.

The reuse frame allows method engineers to drive the ISD crew members to focus on critical aspect(s) of ISD whatever the method evolution is and in this way ensure to always take as much advantage as possible from the method fragmentation and tailoring mechanisms, as it has been previously presented as one of the main goals.

2.2.2 Taking advantage of the reuse frame through ISD

As it has been introduced previously, the reuse frame is used (1) when inserting new method chunks into the repository in order to improve their specification with regards to ISD features and enabling more powerful matching techniques for their retrieval and (2) when searching for method chunks to answer a specific methodological need and to retrieve adequate method chunks in order to build a roadmap dedicated to a specific ISD crew member.

Therefore, means have to be provided to select from the reuse frame the pertinent aspects associated to chunks and methodological need/problem (and its solutions). It is done through the notion of *criterion* and *context*.

A *criterion* is fully defined as a path from the root node *base* to a node n_n of the reuse frame.

$$Cr = [\text{base}, n_1, \ldots, n_n] \text{ with base}, n_1, \ldots, n_n \in RF$$

If n_n is a *family* node, then the exclusion field *exc* must be different from *e* because one of its *aspects* has to be chosen inside the *family*.

$$\text{if type}_{n_n} = \text{family}, \text{exc}_{n_n} \neq e$$

A *context* is defined as a set of *compatible* criteria.

$$Co = \{Cr_1, \ldots, Cr_n\}, \forall Cr_i,\ Cr_j \in Co,\ Cr_i \text{ comp } Cr_j$$

From the previous definitions one can deduce that two criteria are *compatible* if they do not share in their definition a common *family* node n_i with an *exclusion* field equal to *e*.

$$Cr_1 \text{ comp } Cr_2, \forall n_i \in Cr_1 \text{ and } n_j \in Cr_2,\ \text{if } n_i = n_j \text{ then exc}_{ni} \neq e$$

The *method chunk reuse context* is defined as a set of at least one compatible criterion taken from the *reuse frame*.

$$\text{MCC} = \{C_1, \ldots, C_n\}, \forall C_i \in \text{MCC},\ C_i \in RF, \forall C_i, C_j \in \text{MCC},\ C_i \text{ comp } C_j$$

Method chunks providing general guidelines are usually associated to general criteria, that is to say paths ended by nodes from the reuse frame close to the root node. On the contrary, specific guidelines are provided in method chunks associated to precise criteria, that is to say paths ended by nodes from the reuse frame close to aspect nodes or aspect nodes themselves.

It is up to the method engineer who enters the method chunk into the repository to select the most meaningful criteria to qualify the method chunk.

3 Step 1: Assembly-based method construction

The approach for assembly-based method construction aims at building a method 'on the fly' in order to match as well as possible the situation of the project at hand. It consists in selecting reusable method components (that we call method chunks) according to the current project requirements from the method chunks repository and assembling them.

According to Gupta and Prakash [9], method engineering is organized into three main phases: method requirements engineering, method design and method construction and implementation. This approach for assembly-based method construction follows the same way of thinking. As shown in Fig. 10, this approach is requirement-driven, meaning that the project method engineer must start by eliciting requirements for the

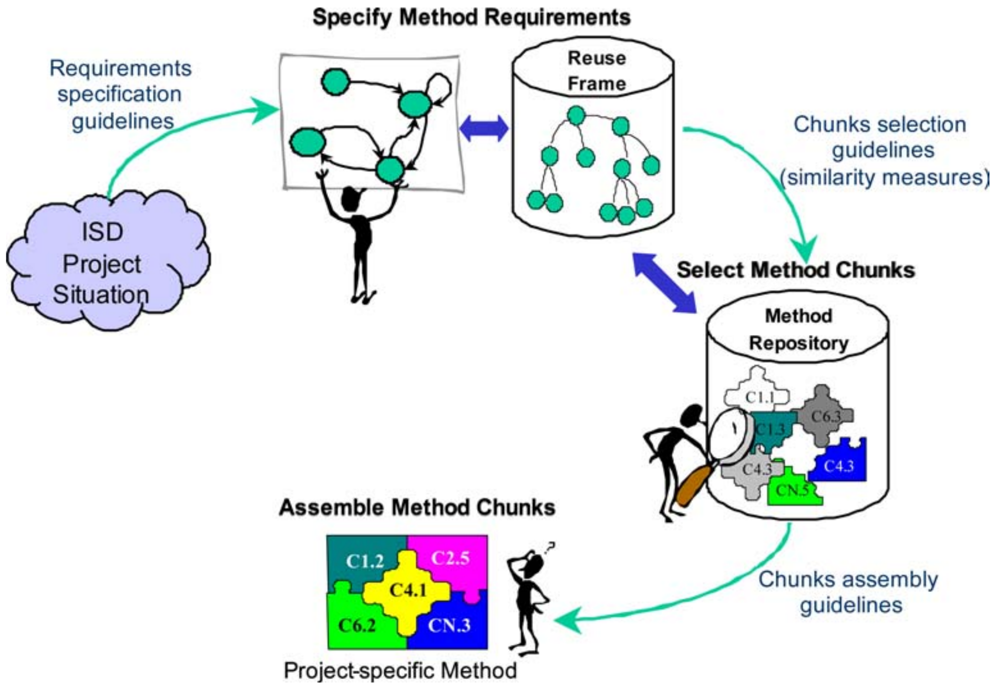


Fig. 10 Overview of the approach for assembly-based method construction

project method. Next, the method chunks matching these requirements can be retrieved from the method chunks repository. And finally, the selected chunks are assembled in order to compose a new method or to enhance an existing one. As a consequence, the three key steps in the assembly-based method engineering process are: *specify method requirements*, *select method chunks* and *assemble method chunks*. This approach provides guidelines supporting the method engineer in each of these steps.

The *requirements specification guidelines* help to define the requirements for the project-specific method in terms of required IS development activities by specifying the type of these activities and the order of their execution. Goal-driven requirements elicitation, scenario-based requirements specification and negotiation, conceptual schema definition, interface modelling are the examples of such activities. The reuse frame is used in order to characterize project situation and to select the predetermined activities.

The *chunk selection guidelines* advise how to retrieve method chunks from the method repository in order to satisfy requirements identified in the previous step. The reuse context of each method chunk is characterized by a set of criteria defined in the reuse frame. Therefore, the reuse frame helps to retrieve method chunks from the method repository.

Finally, the *method construction guidelines* assist the method engineer in the selected method chunks assembly process in order to obtain a homogeneous method. These three steps are detailed in the following sub-sections.

3.1 Method requirements specification

The specification of method requirements for a particular ISD project depends on its initial method situation and the corresponding method engineering (ME) goal. Two core situations can be identified:

1. The IS development crew is used to follow the same method for all ISD projects but considers that, in the current project situations this method must be adapted.
2. The IS development crew does not posses any regular method.

In the first case, the goal of the ME is *to adapt an existing method* whereas in the second case a brand new project-specific method must be constructed.

There are different types of method adaptation. In this approach three of them are distinguished:

– The method in use can be strong in terms of its product model but weak with respect to its process model, which will be the subject of adaptation and enhancement, alternative ways-of-working will be added in the method to its original one.
– The adaptation can be to simply add a new functionality to the existing method, which is relevant in its other aspects.
– Vice versa, the project at hand could not need some functionality offered by the method.

In all these cases, the requirements elicitation process is driven by the identification of the ME intentions such as “add event concept”, “complete completeness checking step”, “replace scenario writing guidelines by the more rich ones” etc., which will allow to complete, enhance or limit the method initially selected. The three adaptation cases can be combined in the selected method adaptation.

In the case of a brand new method construction, the requirements specification is not only to produce the list of ME intentions that will permit to adapt the selected method but to identify the full set of engineering intentions that shall be fulfilled by the new method.

Both of these requirements specification cases lead to a set of requirements expressed as a map (i.e. strategic guideline) [45] that is called the *requirements map* [35]. It can be supposed that the method engineer has to construct a new method supporting: (1) the elicitation of functional system requirements in a goal-driven or actor-driven manner, (2) their conceptualization by using linguistic devices such as scenarios or use cases and (3) their validation in an animated fashion as prototyping and other simulation mechanisms. The map represented in Fig. 11 captures these requirements in three core intentions: "elicit a requirement", "conceptualize a requirement" and "validate a requirement" and imagines several possible strategies to achieve these intentions.

3.2 Method chunks selection

Once the method requirements have been specified, the selection of the method chunks matching these requirements can start. The selection process is based on the requirements map defined in the previous step. The objective that the method engineer has to attain in this step is to select method chunks each satisfying a part of requirements that is covering at least one strategy of the requirements map. Otherwise, a selected method chunk can satisfy all the requirements and cover the whole requirements map.

The chunk selection queries is formulated by specifying the reuse intention of the method chunk and giving values to different reuse context criteria specified in the chunk descriptor [28, 38] as, for example, the system engineering activity in which the chunk is relevant. The situation and intention defined in the chunk interface can also be useful during the chunk selection process. For example, if a method chunk supporting requirements elicitation and specification from initial problem statement, and specifying requirements in terms of goals and/or scenarios is needed, these values should be put in the query as follows:

Reuse_intention.verb = 'Discover' AND Reuse _intention.target = 'Functional system requirements' AND Reuse_context.Engineering activity = ('Requirements elicitation' OR 'Requirements specification') AND Situation = 'Problem statement' AND Intention.Target = ('Goal' OR 'Scenario')

Each retrieved chunk is validated by evaluating its degree of matching to the requirements. This is done by applying similarity measures between the requirements model and the process model of the selected chunk. More details about these similarity measures could be found in [33]. For example, the *L'Ecritoire* method chunk shown in Fig. 7 partly satisfies the requirements captured in the requirements map (Fig. 11). It provides guidelines for functional system requirements elicitation in the form of goal hierarchies and requirements conceptualization in the form of scenarios. The "actor-driven strategy" (Fig. 11) for requirements elicitation can be covered by a method chunk extracted from the *Use case model,* which recommends to identify first the actors of the system under consideration and the role they have to play with the system in order to discover the required use cases. To support animated requirements validation (Fig. 11) one can select method chunks from *RESCUE* method, which uses scenario walkthrough technique for requirements specification and validation [20].

The chunk selection process also provides different guidelines allowing to refine the candidate chunk selection by analysing more in depth if the chunk matches the requirements. For example, if the selected method chunk is an aggregate one, i.e. it is composed of several chunks, the *decomposition guideline* drives the selection of the relevant sub-chunks and the elimination of the inadequate ones. Vice-versa, if the retrieved chunk matches partly the requirements, the *aggregation guideline*

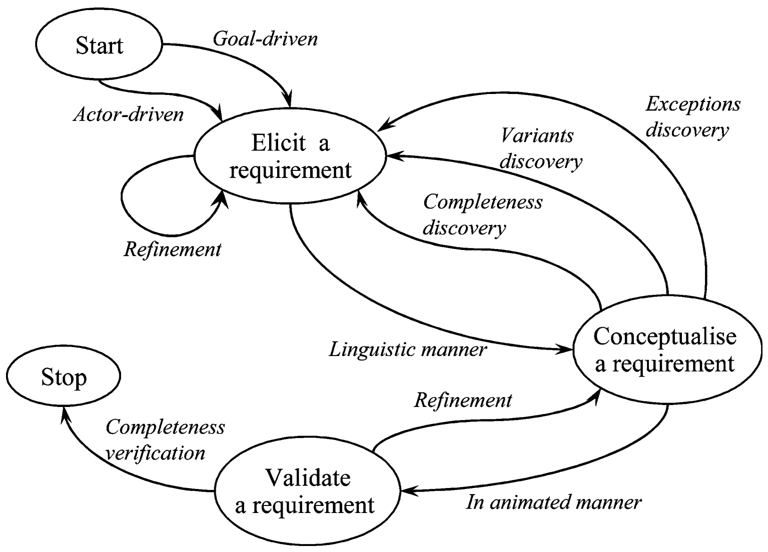


Fig. 11 An example of a requirements map for a new method

proposes to look for an aggregate chunk containing the candidate one based on the assumption that the aggregate chunk might provide a solution for the missing requirements. It is also possible to refine the chunk selection query by modifying selection parameters.

3.3 Method chunks assembly

When at least two chunks have been selected, the method engineer can progress in the assembly of these chunks. We have identified two core types of guidelines in the method chunks assembly process that is called *assembly by association* and *assembly by integration* [33].

The *assembly by association* is relevant when the method chunks to assemble correspond to two different system engineering functionalities, that is, they allow to achieve different engineering intentions and the result of one chunk is used as a source product by the second one. Such method chunks generally do not have common elements in their product and process models and the assembly process is therefore mainly dealing with making the bridge between the two chunks. More precisely, the product models of the chunks must be connected by defining links between their different concepts whereas the connection of their process models consists in defining their execution order.

For example, the method chunk providing guidelines for object life cycle definition requires that the corresponding object class and its relationships with other classes have been identified beforehand. Some information about the events that could occur during the systems life cycle and have some impact on this object class should also be provided. As a consequence, the method chunks for object model construction and events definition must be realized before the chunk for object life cycle definition. The three chunks have complementary objectives. Their assembly is limited to the connection of corresponding concepts as ‘‘class’’, ‘‘state’’ and ‘‘event’’ and the identification of the order in which they must be executed.

The *assembly by integration* is relevant to assemble chunks that have similar engineering goals but provide different ways to satisfy it. In such a case, the process and product models are overlapping, that is, contain some identical or similar elements. The assembly process consists in identifying the common elements in the chunks product and process models and merging them. For example, it could be useful to assemble two method chunks providing different guidelines for requirements elicitation. The chunk supporting goal-driven requirements elicitation combined with the one dealing with actor-driven requirements elicitation allows one to obtain a new chunk providing guidelines that is more rich than the two initial chunks used separately.

Both types of guidelines use a set of assembly operators, operators for product models assembly and operators for process models assembly [36, 37].

For example, the association of two product models is mainly based on the *AddAssociation* and *AddClass* operators: a new association is created between the classes of the initial models in order to connect them into a new model, it can also be done by adding a new class and two associations. On the contrary, the integration of the two product models is based on the application of the *merge* operators as *MergeClasses*, *MergeAssociations,* which allow to connect the initial models by merging similar or identical classes and associations, respectively. *ConnectViaSpecialization* and *ConnetViaGeneralization* operators define respectively the specialization and generalization links between the concepts of the different product models. Their application is useful to build a model of the integrated method chunk that is richer than those of the initial chunks.

In the same manner, the process models of the selected method chunks are assembled: a new transition between the two activities can be added in order to connect two activity-driven models or two activities can be merged into a new one in the case of the overlapping process models integration. If the chunks process models are expressed by using map formalism [45], the *MergeIntentions* and *MergeSections* operators are applied.

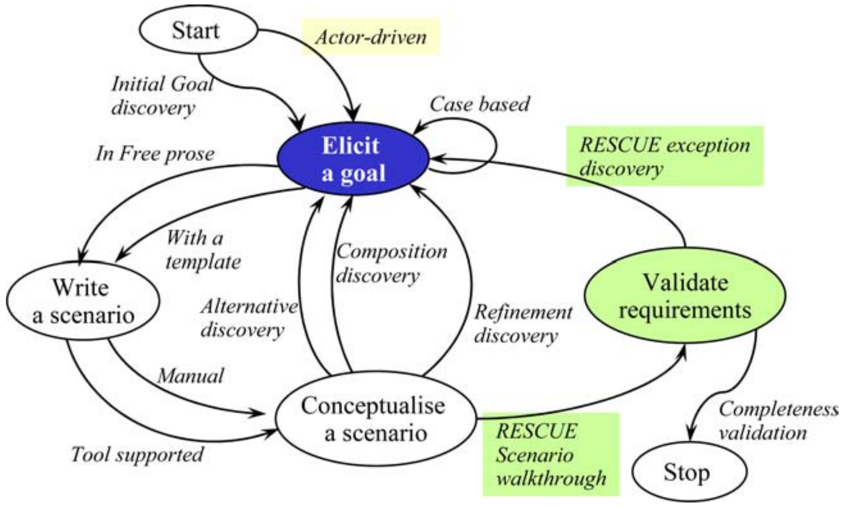


Fig. 12 Example of the project specific method obtained by assembling selected method chunks

Some method chunks adaptation could be required before their assembly in order to avoid concepts name ambiguity. This may be done by applying different *rename* operators as *RenameClass*, *RenameAttribute*, *RenameActivity*, etc., as well as *Retype* operators as *RetypeClass*, *RetypeAttribute*, *RetypeIntention* etc.

The result of the assembly process is a new method that is called as *project-specific method*. This method is an instance of the meta-model for modular methods (Fig. 4). Therefore, it is a collection of tightly coupled method chunks. More exactly, the guideline of this method is a strategic one where each section of its map corresponds to a method chunk or a part of a method chunk.

For example, the selected method chunks (''L'Ecritoire approach'', ''actor-driven use case elicitation'' and ''RESCUE scenario walkthrough technique'') can be assembled by using the integration strategy. These method chunks contain common elements such as ''scenario'' and ''goal'' in their product models and similar intentions such as ''elicit goal'' and ''elicit use case'' in their process models. The integration of the product models consists in merging similar concepts or connecting them via specialization/generalization relationships. In the similar manner, the process models integration consists in merging similar intentions or adding new sections. For example, the method chunk supporting actor-driven use case elicitation is added to the L'Ecritoire map as a new section <Start, Elicit a goal, Actor-driven strategy>. The obtained new method process model is illustrated in Fig. 12.

4 Step 2: Roadmap-driven method configuration

The approach for roadmap-driven method configuration aims at customizing the project-specific method 'on the fly' in order to match the possible profile of the ISD crew member's job within the project at hand. It consists in the selection of method chunks satisfying the ISD crew member's needs within the frame of the project-specific method previously built to answer project requirements.

Indeed customization of ISD methods have mainly been thought for the person in charge of building new methods, that is, the method engineer, in order to allow him/her to construct or adapt a method satisfying the needs of his/her company or the requirements of a specific ISD project. But there is also a need for customization dedicated to each ISD crew member, to provide him or her with guidelines to be followed while performing their daily tasks. ISD crew members also need to benefit, through reuse and adaptation mechanisms, from the experiences acquired during the resolution of previous problems in terms of applying methods in real ISD projects (i.e. using it).

As it has been highlighted during the assembly-based method construction step, different ways may be provided, even inside the same method, to satisfy an engineering goal. Moreover, the sequence through which method chunks have to be used is not always predetermined: they may or not be related by precedence relationships. Therefore, different roadmaps among a set of method chunks are possible. Through this second step of the approach for roadmap driven method configuration different means are provided to find the roadmap, which is the most suitable for the needs of a particular ISD crew member.

The roadmap-driven method configuration is based on three main steps and their underlying techniques: (1) the specification of the methodological problem of the ISD crew member, (2) the retrieval of method chunks matching the problem from the project-specific method built through the first step of the approach and/or from the method chunks repository and (3) the completion of the solution by additional method chunks retrieved from the repository.

In the following each of these techniques are detailed first and then the different possible ways are discussed to use them in order to get a suitable solution to the methodological problem of an ISD crew member.

4.1 Techniques

4.1.1 Specifying the problem of the ISD crew member

First of all, the ISD project crew member must express its problem in terms of applying the project-specific method (resulting from the first step of the approach). The problem is specified through a *problem context* (PC) and eventually a problem intention that the selected method chunks should match. In the problem context, in addition to the pertinent criteria, called *necessary criteria*, one may need to give *forbidden criteria*, that is, the aspects which the ISD project crew member is not interested in. It could be helpful in some cases to be sure the method chunks including these (forbidden) aspects will not appear in the solution.

A PC is defined as:

- *CN*, a set of at least one compatible *necessary criteria* and
- *CF*, a set of compatible *forbidden criteria*.

All criteria are taken from the reuse frame.

$$PC = <CN, CF> \text{ where } CN = \{Cn_1, \ldots, Cn_n\}, \\ CF = \{Cf_1, \ldots, Cf_m\}$$

$$CN \cap CF = \emptyset, \forall Cn_i, Cn_k \in CN, \; Cn_i \text{ comp } Cn_k, \\ \forall Cf_j, Cf_l \in CF, \; Cf_j \text{ comp } Cf_l$$

It is up to the ISD crew member to select the most suitable criteria to qualify his/her situation and need. An ISD crew member may for instance search for chunks dealing with scenario authoring (writing, conceptualizing, validating, etc.). To characterize his/her current working situation he/she may define the problem context through the necessary criteria: ''requirement engineering" (as he/she is working on the requirement engineering phase) and "low time pressure" and through the forbidden criteria: "analysis phase" and "test phase" to be sure that retrieved chunks will be dedicated to requirement engineering.

4.1.2 Selecting method chunks in the project-specific method

Thanks to the method chunk descriptor, the selection from the project-specific method of the method chunks in which the ISD crew member may be interested can be done by matching problem context and method chunk reuse context, as well as problem intention and method chunk reuse intention. Glossaries are provided to specify the intention through meaningful set of verbs, targets and parameters. The *situational metric* (ms) is to introduce, to quantify the matching between the method chunk reuse context and the problem context [26]. This metric is based on (1) the number of common criteria between the necessary criteria (from the problem context) and the method chunk reuse context, (2) the number of common criteria between the forbidden criteria (from the problem context) and the method chunk reuse context, (3) the number of required necessary criteria (from the problem context). It is defined as follows:

$$\mathrm{ms}(mc, pb) = [\mathrm{card}(\mathrm{MCC}_{mc} \cap CN_{pb}) \\ - \mathrm{card}(\mathrm{MCC}_{mc} \cap CF_{pb})]/\mathrm{card}(CN_{pb})$$

where pb is a problem, CN_{pb} the necessary criteria from its *problem context*, CF_{pb} the forbidden criteria from its *problem context*; mc is a method chunk, MCC_{mc} its reuse context.

A positive value indicates that there are more necessary criteria than forbidden ones in the method chunk under consideration with regards to the problem. On the contrary, a negative value indicates that there are less necessary criteria than forbidden ones. The perfect matching is represented by the value 1.

If one defines his/her problem context for instance with "requirement engineering" and "low time pressure" as necessary criteria and "analysis phase" and "test phase" as forbidden criteria, then :

– The chunk matching the problem context only through the "requirement engineering" criterion will get a 0.5 value as situational metric;
– The chunk qualified through the "test phase" and "low time pressure" criteria will get a situational metric equal to 0;
– The chunk characterized only through the "test phase" criterion will get a -0.5 score.

It is a very basic metric that is proposed in order to compute the correspondence between a method chunk and a problem context. As method chunk and problem context contain a small number of criteria (less than 10), this basic measure is easily and rapidly computable.

Thanks to this metric, used in addition to problem intention matching, a set of method chunks corresponding to the ISD project crew member's profile is obtained as a result of the search. Indeed, the selected method chunks may not be related all together: they may for instance cover two disjoint stages of the method (at the beginning and the end of the method for instance) that is, two disconnected sections in the project-specific method map. So, the result set is in fact made of different subsets of organized method chunks.

4.1.3 Searching for method chunks in the repository

Method chunks could be searched directly inside the repository using the reuse intention matching and the situational metrics. It could be useful by itself to search for chunks directly in the repository without looking at the project-specific method. One may for instance search for specific components providing advices on how to build deployment diagram with the help of the UML notation to better understand the purpose of the diagram through its use in the company.

An ISD crew member may also for instance feel a weakness in the project-specific method with regards to his/her specific work. He/she may search directly in the repository to look if he/she finds interesting chunks. In a second time, he/she may ask the method engineer to add the chunk(s) to the project-specific method to enrich it.

When searching for chunks inside the boundary of the project-specific method, the coherency among chunks is ensured by the method itself. When freely searching for chunks inside the whole repository, the coherency of the selected method chunks has to be enforced and means have to be provided to find as much chunks as possible which may satisfy the ISD crew member's needs. In the repository, method chunks are related to each other as aggregated method chunks, *alternative method chunks* or *incompatible method chunks* (Fig. 8). "Elicit a refined goal with refinement discovery strategy" is an example of an aggregate chunk which is composed of two chunks "Elicit a refined goal considering every scenario action as a goal" and "elicit a refined goal using action completion rules" [43]. "Write a scenario in free prose" and "write a scenario with a template" are examples of alternative chunks [44]. A chunk providing advices about object-oriented modelling is not compatible with a chunk embedding guidelines on how to do relational modelling.

Aggregation and alternative relationships may help in enriching the set of retrieved chunks in a coherent way. Incompatibility relationships may help in ensuring the coherency among the retrieved chunks. Degrees are associated to these relationships to respectively quantify the complementarity and the incompatibility. When selecting method chunks directly from the repository, alternative method chunks with high degrees have to be added to the solution and noncompatible method chunks with high degrees have to be removed from the solution. Indeed, the ISD project crew member gives alternativity and incompatibility thresholds to tune the level of coherency he/she wants his/her roadmap to satisfy. A high alternativity threshold will lead to a solution in which only very similar method chunks will be added, while a low one will lead to a solution in which most of them will be included. A high incompatibility threshold will lead to the removal of the very incompatible method chunks from the solution, while a low threshold will lead to the removal of most of the incompatible method chunks.

If the free search in the repository is done to complete the set of chunks embedded in the project-specific method, the result search has to be checked: method chunks which are declared as incompatible with the method chunks assembled in the project-specific method have to be removed from the road-map.

Finally, the set of method chunks still kept in each sequence constitutes the roadmap corresponding to the methodological need of the ISD project crew member.

4.1.4 Tuning facilities

A critical issue in software component reuse is the possibility to augment query when searching the repository [17, 21]. Recent approaches use semantic or natural language-based means to answer this need [31, 50]. In this approach, initial begining from the relationships provided by the application domain and try to use them in order to enlarge the set of chunks retrieved as the answer to a specific need.

The reuse frame introduced previously supports different levels of granularity with regards to criterion definition and use. Criteria with ending node close to the *base* nodes are much more generic than criteria with ending node close to the *aspect* nodes. Indeed, method chunks providing general guidelines are usually associated to general criteria, while specific guidelines are provided in method chunks associated to precise criteria.

When searching for method chunks, one may be interested by method chunks having criteria strictly matching the necessary criteria defined in the problem context. But method chunks with reuse context associated to more specific criteria and usually providing more specific guidelines may also be of interest. They may for instance cover a bigger part of the methodological problem that the ISD crew member has to deal with. In the same way, the ISD crew member may be interested in method chunks having reuse context associated to more general criteria and thus providing more general-purpose guidelines which could also be relevant. One may for instance search for method chunks dealing with “requirements validation” and therefore be also interested in method chunk related to “requirements engineering” in general, which may also deal with validation at some point.

Taking into consideration method chunks with reuse contexts containing more general and/or more specific criteria may also be interesting with regards to the forbidden criteria given in the problem definition. Indeed, enlarging the set of forbidden criteria to more general ones means to forbid full branches of the reuse frame, while enlarging the set of forbidden criteria to more specific ones means to forbid method chunks associated to too specific criteria.

Tuning the selection by allowing or not more general and/or more specific criteria to be included in the necessary and/or forbidden criteria given in the problem definition provides a way for the ISD crew member to reduce or enlarge the number of method chunks included in the solution of his/her problem. If the ISD crew member feels that he/she did not retrieve enough method chunks, he/she may allow more general and/or more specific criteria in order to find more method chunks. On the contrary, if the set of method chunks provided as an answer to his/her problem is too big, he/she may enlarge the set of forbidden criteria by allowing more general and/or more specific criteria and this way cutting branches of the reuse frame and therefore removing from the solution the method chunks associated to these criteria.

4.2 Roadmap building approaches

Based on the techniques introduced (problem specification, method chunk search, selection, and tuning) different ways are proposed to build a roadmap.

The most obvious way to build a roadmap consists in selecting the suitable method chunks inside the project-specific method obtained by applying the first step of the approach. Such a way to build roadmaps is mainly dedicated to neophytes or ISD crew members who want to follow carefully the method built for the project purposes. This way to construct roadmaps is called as *guided roadmap building*.

A much more *free* way consists in searching method chunks directly in the repository without looking at the project method map. Such a way to build roadmaps may be recommended when the ISD crew member is searching for an answer to a very specific problem (leading to the selection of two or three specific chunks).

And finally, a *balanced* way consists in selecting method chunks from the project-specific method and then completing the set of selected chunks by additional ones taken from the repository. Additional chunks could be relevant if they represent alternatives to the chunks embedded in the project-specific method built through the first step of the approach or if they fully answer the problem intention and context given by the project team member. After the application of the chunks from the project-specific method in combination with the additional chunks, the ISD crew member can then give a feedback to the method engineer about his/her experience in applying these additional chunks and ask him/her to assemble some of them into the project method if he/she feels it could be useful for other ISD crew members of this project.

In the following, each of these three ways to build roadmaps are detailed as summarized in Fig. 13.

4.2.1 *Guided roadmap building*

The guided way (Fig. 13) for the roadmap building consists in selecting, in a manual way, method chunks from the project-specific method resulting from the first step of the approach. In other words, the ISD crew member selects different sections in the project-specific method map. It can be supposed that the ISD crew member responsible for system requirements specification will select the method chunks from the project-specific method shown in Fig. 12. Depending on his/her experience, the ISD crew member will select different method chunks. For example, if he/she is novice in goal formalization and scenario writing, he/she will select the method chunk supporting goal elicitation with "actor-driven strategy" instead of "initial goal discovery strategy" and the method chunk for scenario writing "in free prose" instead of "template-based strategy". The first one provides detailed content and style guidelines and is better adapted for the beginners than the second one.

4.2.2 *Balanced roadmap building*

The balanced way (Fig. 13) for the roadmap building consists in four steps:

1. Specifying the ISD problem in terms of intention and context as it has been explained in Sect. 4.1.1.
2. Selecting from the project specific method the method chunks matching the problem intention and problem context with the help of the situational metric and following the tuning parameters given by the ISD crew member, as it has been explained in Sect. 4.1.2.
3. Selecting from the method chunks repository the alternative chunks and chunks matching the problem intention and problem context as it has been explained in Sect. 4.1.3, including consistency check between the method chunks from the project-specific method and the method chunks from the repository. Tuning facilities may be used through these two last steps.
4. Finally, the ISD crew member may proceed to an ultimate selection in a manual way, as suggested in the guided building way.

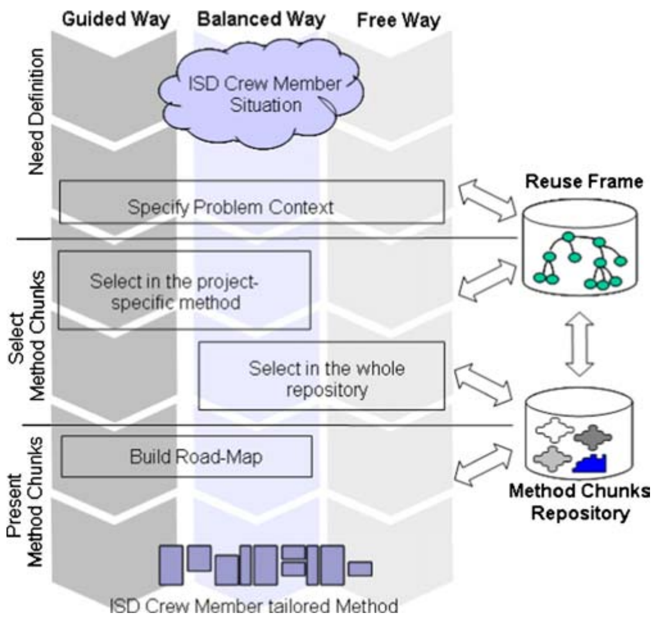


Fig. 13 Overview of the roadmap-driven method configuration approach

For example, instead of using the method chunk for requirements validation proposed by the project specific-method Fig. 12 (the RESCUE Scenario walkthrough technique to validate requirements), the ISD crew member can select another method chunk from the method chunks repository supporting requirements validation in a different manner, for instance, some prototyping technique.

4.2.3 *Free roadmap building*

This last way to build roadmaps consists in directly selecting method chunks from the repository without the help of the project-specific method. Therefore, such a process starts by specifying the ISD problem in terms of intention and context as it has been explained in Sect. 4.1.1. Then, the search is processed directly in the repository with the help of the situational metrics and following the tuning parameters given by the project team member, as it has be explained in Sects. 4.1.2 and 4.1.4.

5 Conclusion

In this paper two perspectives of SME are considered: (1) project-specific ME, which aims to satisfy specific ISD project method requirements and (2) engineer-specific ME dedicated to satisfy requirements of an individual ISD crew member. It is believed that these two perspectives are complementary and should be combined in order to support the ISD process at hand.

Therefore, in this work two SME approaches are combined, the assembly-based approach for project-specific method construction and the roadmap-driven approach for engineer-specific method configuration, as two core steps in the global SME approach. Even though, each of these approaches has been defined previously, independently, their integration is possible thanks to the concept of method chunk shared by both of them. A method chunks repository represents the basis of each of the two initial approaches. Besides, the reuse frame is another common element in both of these two ME approaches.

The approach for assembly-based method construction, which represents the first step in the global SME approach, focuses on project-specific method building. This approach provides guidelines for the specification of method requirements in line with the situation of the project at hand, the selection of the method chunks satisfying these requirements and the assembly of the selected method chunks. The method requirements specification is based on the criteria defined in the reuse frame and allowing to characterize

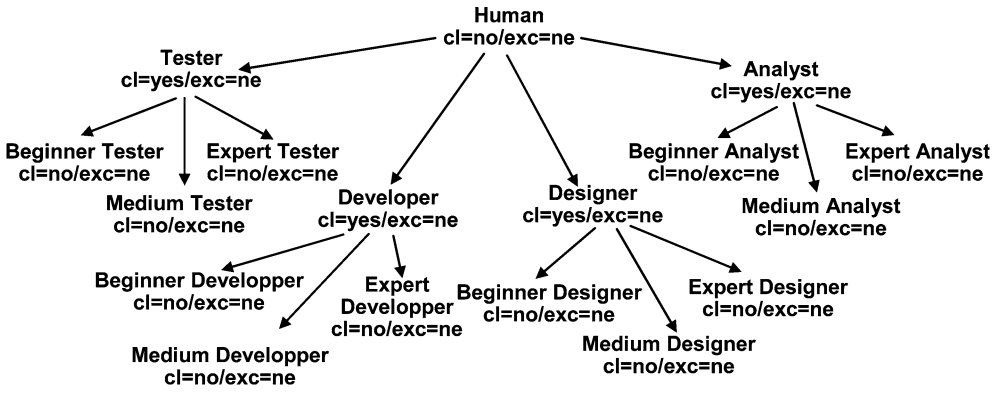


Fig. 14 Reuse frame: human

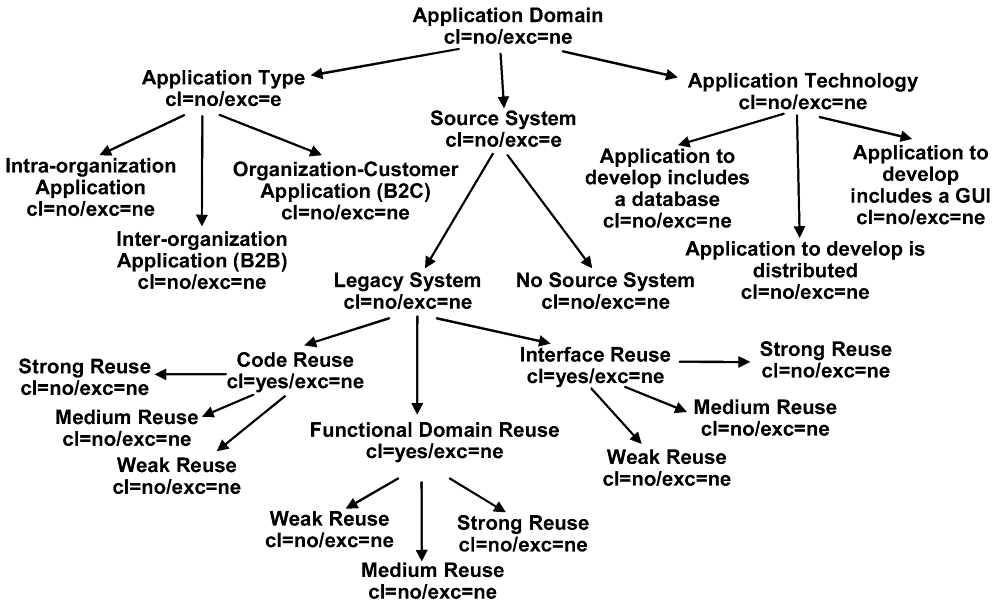


Fig. 15 Reuse frame: application domain

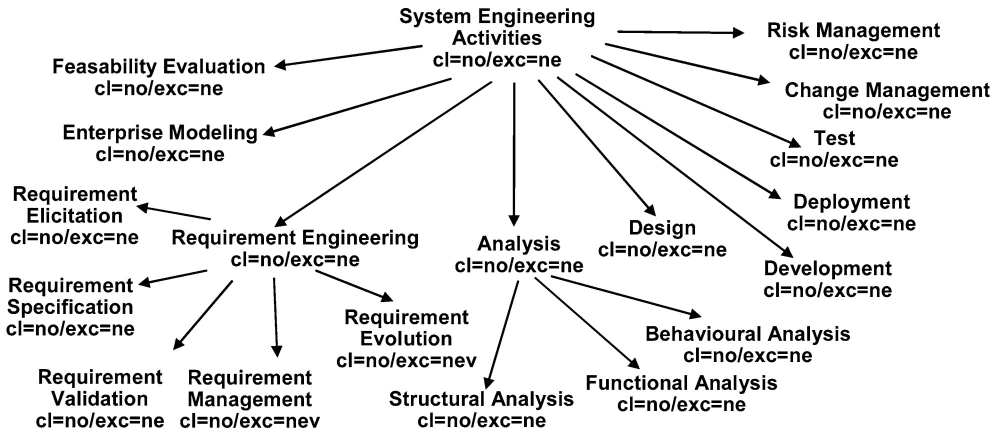


Fig. 16 Reuse frame: system engineering activities

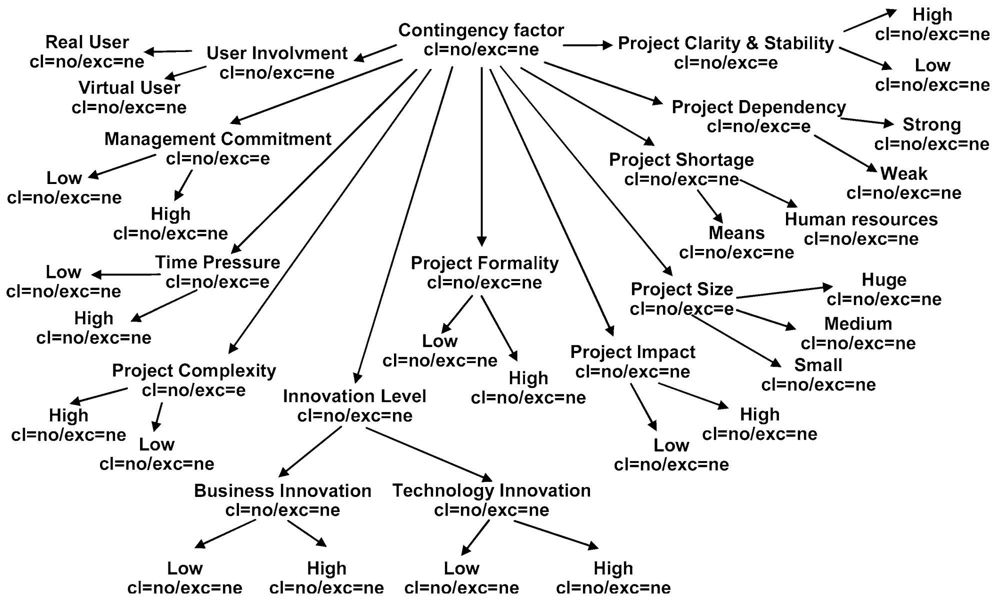

Contingency factor cl=no/exc=ne
User Involvment cl=no/exc=ne
Real User cl=no/exc=ne
Virtual User cl=no/exc=ne
Management Commitment cl=no/exc=e
Low cl=no/exc=ne
High cl=no/exc=ne
Time Pressure cl=no/exc=e
Low cl=no/exc=ne
High cl=no/exc=ne
Project Complexity cl=no/exc=e
High cl=no/exc=ne
Low cl=no/exc=ne
Innovation Level cl=no/exc=ne
Business Innovation cl=no/exc=ne
Technology Innovation cl=no/exc=ne
Low cl=no/exc=ne
High cl=no/exc=ne
Low cl=no/exc=ne
High cl=no/exc=ne
Project Formality cl=no/exc=ne
Low cl=no/exc=ne
High cl=no/exc=ne
Project Impact cl=no/exc=ne
Low cl=no/exc=ne
High cl=no/exc=ne
Project Size cl=no/exc=e
Huge cl=no/exc=ne
Medium cl=no/exc=ne
Small cl=no/exc=ne
Project Shortage cl=no/exc=ne
Means cl=no/exc=ne
Human resources cl=no/exc=ne
Project Dependency cl=no/exc=e
Strong cl=no/exc=ne
Weak cl=no/exc=ne
Project Clarity & Stability cl=no/exc=e
High cl=no/exc=ne
Low cl=no/exc=ne

Fig. 17 Reuse frame: contingency factors

the situation of the project at hand. The reuse frame is also used during the method chunks selection process together with the product end process elements glossary in order to support the construction of queries. The assembly guidelines propose a set of assembly operators allowing to assemble product and process parts of the selected method chunks. Moreover, the similarity measures are provided in order to detect overlapping method chunks as well as to compare the method requirements with the solutions proposed by the selected chunks.

The approach for roadmap-driven method configuration represents the second step in the global SME approach and is dedicated to personalize the project-specific method defined previously for each ISD project crew member in order to support his/her specific tasks. Following this approach, the method chunks are selected from the project-specific method (defined during the first step) according to the requirements of a particular engineer and represent the corresponding engineer's roadmap. If it is necessary, additional method chunks can be retrieved from the method chunks repository to complete this roadmap. Finally, in some specific cases, it could be necessary to select method chunks only from the repository without using project-specific method. Guidelines are provided to support the

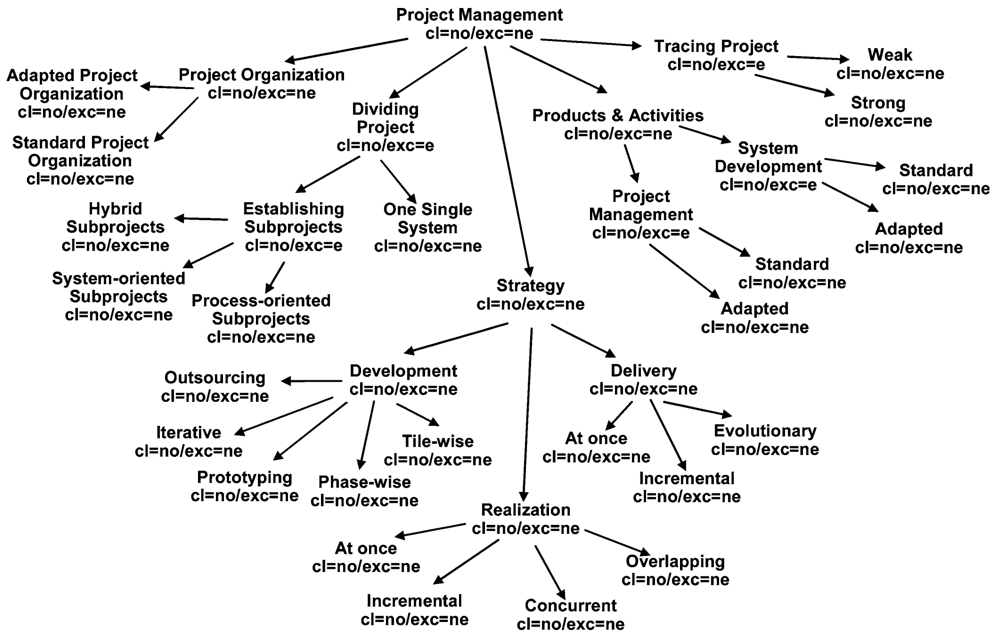

Project Management cl=no/exc=ne
Project Organization cl=no/exc=ne
Adapted Project Organization cl=no/exc=ne
Standard Project Organization cl=no/exc=ne
Dividing Project cl=no/exc=e
Establishing Subprojects cl=no/exc=e
Hybrid Subprojects cl=no/exc=ne
System-oriented Subprojects cl=no/exc=ne
Process-oriented Subprojects cl=no/exc=ne
One Single System cl=no/exc=ne
Tracing Project cl=no/exc=e
Weak cl=no/exc=ne
Strong cl=no/exc=ne
Products & Activities cl=no/exc=ne
System Development cl=no/exc=e
Standard cl=no/exc=ne
Adapted cl=no/exc=ne
Project Management cl=no/exc=e
Standard cl=no/exc=ne
Adapted cl=no/exc=ne
Strategy cl=no/exc=ne
Development cl=no/exc=ne
Outsourcing cl=no/exc=ne
Iterative cl=no/exc=ne
Prototyping cl=no/exc=ne
Phase-wise cl=no/exc=ne
Tile-wise cl=no/exc=ne
Delivery cl=no/exc=ne
At once cl=no/exc=ne
Incremental cl=no/exc=ne
Evolutionary cl=no/exc=ne
Realization cl=no/exc=ne
At once cl=no/exc=ne
Incremental cl=no/exc=ne
Concurrent cl=no/exc=ne
Overlapping cl=no/exc=ne

Fig. 18 Reuse frame: project management

three ways for roadmap construction called guided, balanced and free way, respectively.

The assembly-based project-specific method building asks for a considerable knowledge about ISD methods, meta-modelling, method chunks assembly as well as capability to evaluate the situation of the entire project and its method requirements. Therefore, this step should be realized by the method engineer or the manager of the current project. On the contrary, the approach for roadmap-driven method configuration can be executed by each ISD crew member.

It is evident, that the obtained global SME approach is richer than the two initial ones as it allows to cover a large scale of SME requirements. To evaluate this approach by applying it in real ISD projects is the current preoccupation. Two prototypes, one for each ME approach, have been developed. For the moment they are really simple and only support method chunks storage and selection but not assembly processes. Our current objective is to develop software environment integrating these two approaches.

Besides, it is thought that the ME ontology should be richer than the reuse frame and glossary. This ontology is necessary to facilitate method chunks characterization and formalization, to permit their comparison, to position them in the method chunks repository, to define different relationships between method chunks, and so on.

6 Annexe: Reuse frame

As it has been presented in this paper, the reuse frame is a polymorphic structure allowing to do faceted classification of reusable assets dedicated to method engineering. In Section 2.2.1 explanation was given on how one came to the conclusion that IS critical aspects may always be classified into three main topics: *application domain*, *organizational* and *human*. Starting from this decomposition, each company may populate the reuse frame with its own relevant criteria in order to let ISD crew members adapt the project or company-specific method in a meaningful way for the company. But in this annexe, a reuse frame content is also provided that built from various works made on meaningful criteria with regards to method customization [24, 49]. Figures 14 and 15 show the *human* branch and the *application domain* branch respectively. In Figs. 16 and 17 *system engineering activities* and *contingency factors* are presented. Finally, Fig. 18 shows the aspects associated to the *project management* family. The reuse frame provides qualitative information about meaningful criteria for method customisation. Therefore, scales have been used instead of numeric values to qualify aspects. An effort was made to reduce the number of fillers in a scale as much as possible in order to have a clear and easily usable set of aspects. When dealing with large scales (i.e. scales including a lot of fillers) it becomes difficult to select the right filler inside the scale in order to qualify a method chunk. For example, it is more difficult to allocate the right filler to an aspect if its scale is very detailed as {"very small", "small", "medium", "big", "very big", "huge"} in comparison if it is only composed of three fillers {"small", "medium", "big"}.

References

1. The American Heritage Dictionary of the English Language (2000) Houghton Mifflin Company
2. Bajec M, Vavpotic D, Kirsper M (2004) The scenario and tool-support for constructing flexible, people-focused system developement methodologies. In: Proceedings of the international conference on information systems development, ISD 2004. Vilnius, Lithuania
3. Brinkkemper S, Saeki M, Harmsen F (1998) Assembly techniques for method engineering. In: Proceedings of the 10th international conference on advanced information systems engineering, CAiSE'98. Pisa, Italy, LNCS 1413. Springer, Berlin Heidelberg New York
4. Cauvet C, Rosenthal-Sabroux C (2001) Ingénierie des systèmes d'information. Hermes
5. Deneckère R, Souveyet C (1998) Patterns for extending an OO model with temporal features. In: Proceedings of the international conference on object-oriented information systems, OOIS'98. Springer, Paris, France
6. Firesmith DG, Henderson-Sellers B (2002) The OPEN process framework—an introduction Addison-Wesley, Harlow
7. Fitzgerald B (1997) The use of systems development methodologies in practice: a field study. Inf Sys J 7(3):201–212
8. Graham I, Henderson-Sellers B, Younessi H (1997) The OPEN process specification. Addison-Wesley, Harlow
9. Gupta D, Prakash N (2001) Engineering methods from method requiremnets specifications. Requirments Eng J 6(3):135–160
10. Harmsen AF (1997) Situational method engineering. Moret Ernst & Young, Utrecht
11. Harmsen AF, Brinkkemper S, Oei H(1994) Situational method engineering for information system projects. In: Olle TW, Verrijn Stuart AA (eds) Methods and associated tools for the information systems life cycle, Proceedings of the IFIP WG 8.1 working conference CRIS '94. North-Holland, Amsterdam, pp 169–194
12. Iivari J, Maansaari J (1998) The usage of systems development methods: are we stuck to old practice? Inf Sys Technol 40(9):501–510
13. Introna LD, Whitley EA (1997) Against methodism: exploring the limits of method. Inf Technol People 10(1):31–45
14. Jacobson I, Christenson M, Jonsson P, Oevergaard G (1992) Object oriented software engineering a use case driven approach. Addison-Wesley, Harlow
15. Jarke M, Rolland C, Sutcliffe A, Domges R (1999) The NATURE requirements Engineering. Shaker Verlag, Aachen
16. Karlsson F, Ågerfalk PJ (2004) Method configuration: adapting to situational characteristics while creating reusable assets. Inf Softw Technol 46(9):619–633
17. Khayati O (2002) Components retrieval systems reuse in object-oriented information systems design. OOIS workshop, Montpellier
18. Kumar K, Welke RJ (1992) Method engineering, a proposal for situation-specific methodology construction. In: Cotterman W, Senn J (eds) Systems analysis and design: a research agenda. Wiley, New York, pp 257–268
19. Lings B, Lundell B (2004) Method-in-action and method-in-tool: some implications for case. In: Seruca I et al (eds) Proceeedings of the 6th international conference on enterprise information systems (ICEIS 2004). Insticc Press, pp 623–628
20. Maiden NAM, Jones SV, Manning S, Greenwood J, Renou L (2004) Model-driven requirements engineering: synchronising

models in an air traffic management case study, Proceedings CAISE'04. Springer, Berlin Heidelberg New York, LNCS 3084, pp 368–383

21. Mili H, Valtchev P, Di-Sciullo A, Gabrini P (2001) Automating the indexing and retrieval of reusable software components. In: Proceedings of the 6th international workshop NLDB, June 28–29, Madraid, Spain, pp 75–86
22. Mirbel I, de Rivieres V (2002) Adapting analysis and design to software context: the JECKO approach. In: Proceedings of the international conference on object-oriented information systems (OOIS'02), Montpellier, France, pp 223–22
23. Mirbel I, de Rivieres V (2003) Conciliating user interface and business domain analysis and design. In: Proceedings of the 9th international conference on object-oriented information systems, OOIS'03, Geneva
24. Mirbel I, de Rivieres V (2003) Towards a UML profile for building on top of running software. In: UML and the Unified Process. IRMA Press, Hershey
25. Mirbel I (2004) A polymorphic context frame to support scalability and evolvability of information system development processes. In: Proceedings of the international conference on enterprise information systems, ICEIS'04, Porto, Portugal
26. Mirbel I (2004) Rethinking ISD methods: fitting project team members profiles. I3S technical report I3S/RR-2004-13-FR
27. Nilsson A, (2004) Information systems development—past, present and future trends. In: Invited talk in the international conference on information systems development, ISD'04. Vilnius, Lithuania
28. Plihon V, Ralyté J, Benjamen A, Maiden NAM, Sutcliffe A, Dubois E, Heymans P (1998) A reuse-oriented approach for the construction of scenario based methods. In: Proceedings of the international software process association's 5th international conference on software process (ICSP'98), Chicago, Illinois
29. Prakash N (1999) On method statics and dynamics. Inf Syst 34(8):613–637
30. Prat N (1997) Goal formalisation and classification for requirements engineering. In: Proceedings of the 3rd international workshop on requirements engineering: foundations of software quality REFSQ'97, Barcelona, pp 145–156
31. Pujalte V, Ramadour P (2004) Réutilisation de composants: un processus interactif de recherche. Majecstic'05 Calais
32. Punter HT, Lemmen K (1996) The MEMA model: towards a new approach for method engineering. Inf Softw Technol 38(4):295–305
33. Ralyté J, Rolland C (2001a) An assembly process model for method engineering. In: Proceedings of the 13th international conference on advanced information systems engineering (CAISE'01), Interlaken, Switzerland, LNCS 2068. Springer, Berlin Heidelberg New York, pp 267–283
34. Ralyté J, Rolland C (2001b) An approach for method reengineering. In: Proceedings of the 20th international conference on conceptual modeling (ER2001), Yokohama, Japan, LNCS 2224. Springer, Berlin Heidelberg New York, pp 471–484
35. Ralyté J (2002) 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'02). Kluwer, Kanazawa, pp 127–152
36. Ralyté J, Rolland C, Deneckère R (2004) Towards a Meta-tool for change-centric method engineering: a typology of generic operators. In: Proceedings of the 16th international conference CAISE'04, Riga, Latvia. Springer, Berlin Heidelberg New York
37. Ralyté J, Rolland C, Plihon V (1999a) Method enhancement with scenario based techniques. In: Proceedings of the 11th international conference on advanced information system engineering (CAISE'99), LNCS 1626. Springer, Berlin Heidelberg New York, Germany, pp 103–118
38. Ralyté J (1999b) Reusing scenario based approaches in requirement engineering methods: CREWS method base. In: Proceedings of the 10th international workshop on database and expert systems applications (DEXA'99), 1st international workshop on the requirements engineering process—innovative techniques, models, tools to support the RE process (REP'99), Florence, Italy. IEEE Computer Society, pp 305–309
39. Rolland C, Plihon V (1996) Using generic chunks to generate process models fragments. In: Proceedings of the 2nd IEEE international conference on requirements engineering, ICRE '96, Colorado Spring
40. Rolland C, Prakash N (1996) A proposal for context-specific method engineering. In: Proceedings of the IFIP WG 8.1 conference on method engineering. Chapman & Hall, Atlanta, pp 191–208
41. Rolland C, Nurcan S, Grosz G (2000) A decision making pattern for guiding the enterprise knowledge development process. J Inf Softw Technol 42:313–331
42. Rolland C, Plihon V, Ralyté J (1998) Specifying the reuse context of scenario method chunks. In: Proceedings of the 10th international conference on advanced information system engineering (CAISE'98), Pisa, Italy, LNCS 1413. Springer, Berlin Heidelberg New York, pp 191–218
43. Rolland C, Souveyet C, Ben Achour C (1998a) Guiding goal modelling using scenarios. IEEE Trans Softw Eng 24(12):1055–1071
44. Rolland C, Ben Achour C (1998b) Guiding the construction of textual use case specifications. Data Knowl Eng J 25(1):125–160
45. Rolland C, Prakash N, Benjamen A (1999) A multi-model view of process modelling. Requirements Eng J 4(4):169–187
46. Rossi M, Ramesh B, Lyytinen K, Tolvanen J (2004) Managing evolutionary method engineering by method rationale. J Assoc Inf Syst 5(9):356–391
47. Saeki M, Iguchi K, Wen-yin K, Shinohara M, (1993) A meta-model for representing software specification & design methods. In: Proceedings of the IFIPWG8.1 conference on information systems development process, Come, pp 149–166
48. van Slooten K, Brinkkemper S (1993) A method engineering approach to information systems development". In: Prakash N, Rolland C, Pernici B (eds) Information systems development process. Elsevier, North-Holand, pp 167–186
49. van Slooten K, Hodes B (1996) Characterizing IS development projects. In: Proceedings of the IFIP WG 8.1 conference on method engineering. Chapman and Hall, pp 29–44
50. Sugumaran V, Storey VC (2003) A semantic-based approach to component retrieval. The database for advances in information systems, vol.34, No 3
51. Tawbi M, Souveyet C, Rolland C (1998) L'ECRITOIRE a tool to support a goal-scenario based approach to requirements engineering. Inf Softw Technol J
52. Object Management Group (2004) Unified modelling language (UML), version 1.5. Available at http://www.omg.org/technology/documents/formal/uml.htm