# A POLYMORPHIC CONTEXT FRAME TO SUPPORT SCALABILITY AND EVOLVABILITY OF INFORMATION SYSTEM DEVELOPMENT PROCESSES

Isabelle Mirbel
*I3S Laboratory*
*Les Algorithmes*
*Route des Lucioles*
*BP 121*
*06903 Sophia Antipolis Cedex*
*FRANCE*

Keywords: Information System, Method Engineering, Fragment, Reuse, Evolution

Abstract: Nowadays, there is an increasing need for flexible approaches, adaptable to different kinds of Information System Development (ISD). But customization of ISD processes have mainly be thought of for the person in charge of building processes, i.e. the methodologists, in order to allow him/her to adapt the process to the need of its company or projects. But there is also a need for customizations dedicated to project team members (application engineers), to provide them with customized guidelines (or heuristics) which are to be followed while performing their daily task. The knowledge capitalization framework we propose supports evolvability and customization of ISD processes. Reuse and customization are handled through *process fragments* stored in a dedicated *repository*. Our purpose is not to propose a new way to built processes, as several approaches already exist on this topic, but to ease the use of existing ones by making them less rigid and allowing their adaptation to the need of the company, the project and most of all, the project team member. Therefore, in addition to a *repository* of *process fragments*, we propose a scalable and polymorphic structure allowing methodologists to define a working space through a *context* made of *criterias*. Thanks to this *context* the project team members better qualify their ISD problem in order to find a suitable solution. A solution is made of *process fragments* organized into a *route-map* specially built to answer the project team member need and directly usable by him/her.
The *context-frame* we focus on in this paper is a scalable structure which supports evolution and tailoring by the methodologists for the project team member's need with regards to project and process features.

## 1 INTRODUCTION

Information System Development (ISD) is different each time, depending on the situation and context. A given technique, notation or mechanism may be used in a different way depending on the development under consideration. A process (or method) that has proved its power for certain kinds of development may be quite unsuitable for others. There is no universal applicability of processes (Ralyte, 2001). Therefore, there is a need for flexible approaches, adaptable to different kinds of development. The need for situation-specific processes, to better satisfy particular situation requirements, has already been emphasized (van Slooten and Hodes, 1996; Brinkkemper et al., 1998).

Moreover, there is an increasing need for lightweight processes by opposition to heavyweight ones. Lightweight processes increase project team members (or application engineers) involvement on the contrary of heavyweight processes where the only significant choice is made by methodologists who chose the development process. Lightweight processes focus on finding the best way for the current situation. Project team members choose from as many alternative paths as possible.

Indeed customization of ISD processes (Ralyte, 2001; Brinkkemper et al., 1998) have mainly be thought of for the person in charge of building a new process, i.e. the methodologists, in order to allow him/her to adapt the process to the need of its company or projects. But there is also a need for customizations dedicated to project team members, to provide them with guidelines (or heuristics) which are to be followed while performing their daily task (Gnatz et al., 2001; Mirbel and de Rivieres, 2002b). The adaptation is handled through the breaking down of guidelines (heuristics) into fragments, which may then be selected if they answer to the project team member need.

Mirbel I. (2004).
A POLYMORPHIC CONTEXT FRAME TO SUPPORT SCALABILITY AND EVOLVABILITY OF INFORMATION SYSTEM DEVELOPMENT PROCESSES.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 131-138
DOI: 10.5220/0002608101310138

It is recognized as important to benefit from the experiences acquired during the resolution of previous problems through reuse and adaptation mechanisms (Cauvet et al., 2001). With regards to software development, reuse has been widely studied from the product point of view (Gamma et al., 1995; Fowler, 1997), but it is now also a challenging issue to handle it from the process point of view.

Moreover, the constant evolution of techniques, mechanisms and technologies provided to support ISD requires evolution-oriented development processes. Adapted processes have to be developed to take advantage of new technologies. Real world evolves and implies changes of the supporting information system. Handling correctly information system evolution also means to use appropriate ISD processes.

We propose a knowledge capitalization framework to support evolvability of ISD processes. Reuse and customization are handled through *process fragments* stored in a dedicated *repository*. This framework is mainly dedicated to project team members and allow them to let others benefit from their experience in solving ISD problem by storing their solution in terms of fragments inside the repository. Our framework also allows them to retrieve fragments corresponding to their ISD problem by using the *process fragment repository*. The key element of such a repository is the means proposed to store and retrieve fragments.

From the methodologist point of view, we believe capitalization could be much more useful if driven to focus on critical aspects of development process ; and customization much more efficient if kept inside the boundaries of the company development process.

To answer this twofold need (fragment manipulation means for project team members and process control for methodologists), we propose a scalable and polymorphic structure, the *context frame*. It helps project team members to specify fragments in a way anticipating their reuse, and to well express their ISD problem to find a suitable solution. The *context frame* can be seen as an ontology dedicated to ISD processes. Ontology for development processes is a current high topic of work in the field of method engineering (Saeki, 2003). Our *context frame* is managed by methodologists allowing them both to drive the project team members on critical aspects when creating and retrieving fragments and to keep customization (supported by fragment selection) inside the boundaries of the company development process. The *context frame* is not a fixed structure and evolves through the time. Its content is controlled by the methodologists, giving them a way to support the evolvability of development processes and project needs. It is the purpose of this paper to present this *context frame*.

We start first by presenting the whole framework in section 2. Then, the *context frame* is described in section 3. The different kinds of context required to support scalability in ISD processes are first presented in section 4. Then, in section 5, their usefulness for ISD *by* reuse, as well as for ISD *for* reuse is discussed. Finally, we conclude in section 6.

## 2 A FRAMEWORK TO CAPITALIZE KNOWLEDGE ABOUT ISD PROBLEM SOLVING

During ISD, heuristics are elaborated and may be useful to other teams facing close situations in different projects independently of the functional domain as well as the technical domain. Our approach allows to reassemble heuristics accumulated by project team members to help focusing on critical aspects of development and to take advantage of the way to solve problems. We focus on the re-use of the way of working, on the way to apprehend ISD tasks.

Our approach aims at guiding project team members to most appropriately apply a set of techniques and methods so as to focus on critical aspects of ISD in order to better handle its complexity. Therefore, we propose a *context frame* which allow methodologists to define the working context and project team members to situate their work with regards to this context. The *context frame* is managed by the methodologists and allows them to set the boundaries inside which customization will be possible with regards to the project team members needs. It allows methodologists to provide customized processes and to keep project team members inside the framework of the process used by the company. Project team members use the *context frame* defined by methodologists to characterize the new *fragments* they introduce in the repository. By situating their fragment with regards to the criterias of the context frame, they anticipate their reuse in the framework of the company development process. Project team members also express their ISD *problem* with the help of the *context frame* to select and reuse fragments from the repository through ISD *by* reuse.

The main goal of the *process fragment repository* is to help project team members through their daily tasks. Most of the time, ISD processes are defined with regards to the phase they are involved in (Henderson-Sellers and Edwards, 1990; Boehm, 1988; Royce, 1970), with regards to the results to be obtained (Finkelstein et al., 1990; Franckson and Peugeot, 1991). But to get a valuable result, it is not enough to use the dedicated diagram(s) and concept(s) at the right moment. It is also necessary for

the project team members to understand what he/she is working on and why he/she is using such a diagram or concept has to be taken into account (Potts, 1989; Rolland and Souveyet, 1995; si Said and Rolland, 1998). In our approach, ISD tasks are driven by the problem one may face through his/her daily work, instead of the result he/she wants to get.

Notations provided to support ISD tasks are more and more richer (Object Management Group, 2001). They also become more and more complex and therefore seems to be the heart of ISD: project team members focus on the use of the different diagrams or concepts provided by the notation much more than on the tasks to be performed. Our approach helps in re-centering the work on tasks instead of notation.

With regards to ISD *for* reuse, *fragments* are defined through an *intention* inciting them to carefully specify the aim of the fragment. A *fragment context* is associated to each new-built fragment to enforce its definition with regards to the working framework.

With regards to ISD *by* reuse, requests are expressed in terms of *problem* and *problem context*. *Solutions* are given in terms of *route-map* made of *fragments* including guidelines and heuristics to help in ISD tasks.

An example of fragment is given in figure 1. In this figure, a fragment named `Requirement-out-of-scope` is presented. Its *fragment context* indicates it presents guidelines for when dealing with a running software. There is an associate fragment `DB-out-of-scope`, which is complementary. There is no incompatible fragments. The intention explain the purpose of the fragment which is to help in documenting the running part of the application under which the development will take place. Associated guidelines in UML are then given.

| | |
|---|---|
| **Name** | **Requirement–Out–of–Scope** |
| **Fragment Context** | {[base – software – running software]} |
| **Related Fragments** | BusinessDomain–Out–of–Scope – complementarity – 0.75 |
| **Non–compatible Fragments** | – |
| **Intention** To document existing parts of the running software useful to understand the purpose of the new software but not directly related to the new development. | |
| **Guidelines** | **Notation** UML use–case diagrams |
| | **Description** Add use–cases dedicated to running functionalities increasing the understanding of the software. Stereotype them with <<out–of–scope>>.<br>Group all the use–cases stereotyped <<out–of–scope>> in a package (or set of packages) also stereotyped <<out–of–scope>> |

Figure 1: An example of fragment

Figure 2 summarizes the structure of our framework with the two main sources of information: the *context frame* and the *process fragment repository*, on top of which mechanisms for ISD *by* re-use and *for* re-use are provided to the project team members, while means of context management are provided to methodologists.

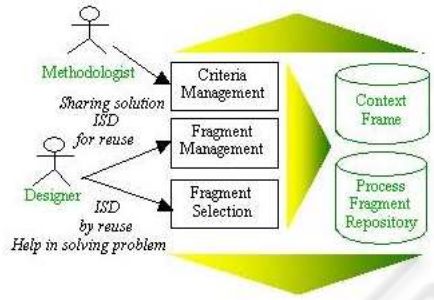


Figure 2: Framework overview

In this paper, we focus on the *context frame*. And we show how useful it is for ISD *by* reuse and *for* reuse.

# 3 A SCALABLE AND POLYMORPHIC CONTEXT FRAME

The *context frame* allows methodologists to define a common working frame for all the project team members and all the projects in the company. The working frame is defined through a *context* made of *criterias* and allowing then project team members to better qualify their ISD problem in order to get a customized solution.

A criteria represents a point of view or an aspect on the development process that methodologists want to highlight in order to help project team members to apply techniques and methods to focus on critical aspects of the development process. Through ISD *for* reuse, criterias help in qualifying fragments to anticipate and ease their reuse. Through ISD *by* reuse, criterias help in specifying the problem to solve in order to get adapted fragments. Information characterizing the software to develop (*Software to develop includes a* `User Interface`, *Software to develop includes a* `database`, *Software to develop is* `distributed`, *Software to develop is build on a* `running` *application*), the competence of the project team members in charge of the development (*expert, medium, beginner*), and the project management features (*Delivery strategy, Realization strategy, Time pressure*) are example of useful criterias to customize the process.

In the following, we discuss first the structure of the *context frame*. Then, we show how to use it.

# 4 CONTEXT FRAME DEFINITIONS

A *context frame* is a tree which root-node is the most generic criteria called *Base*. Leaf-nodes correspond to *criterias* or *families* of criteria.

- A *criteria* is a leaf node in the tree. It is described by its *name*. Examples of criterias are *software to develop includes a* `database`, or *guidelines dedicated to* `expert` *designers*.
- A *family* of criteria is a non-leaf node with at least two sub-nodes (which could be *criteria* or *family* nodes). A *family* is described by a *name* and information about relationships among the different criterias or sub-families constituting the current family. Families are interesting to allow a better understanding of criterias entered in the framework, from the project team member point of view as well as from the methodologist point of view. For instance, as we deal with different criterias characterizing the software to be developed: *Software to develop includes a* `User Interface`, *Software to develop includes a* `database`, *Software to develop is* `distributed`, *Software to develop is build on a* `running` *application*, we can group all these criterias into a family called `Software to develop`. This non-leaf node helps methodologists to maintain the context frame and project team members to understand and use the grouped criterias.

  The objective of the context frame is to help:

  – through ISD *for* reuse by providing a means to organize fragments and
  – through ISD *by* reuse by providing a means to select the right fragments to build a *solution* corresponding to a *problem*.

  Therefore, additional information aiming at better specifying the way criterias belong to a family helps in using them to constitute a coherent context. Two kinds of information are provided:

  – The *order* field indicates if direct criterias or sub-families are ordered ($ord = o$) or not ($ord = no$). For instance, if `development process phase` is a *family*, `analysis`, `design`, `implementation` and `test` are criterias which are ordered (`analysis` is processed before `design`, which is processed before `implementation`, which is processed before `test`). It is interesting to indicate it because when retrieving fragments associated with the `design` criteria for instance, it may also be interesting to look at fragments associated with the `analysis` and `implementation` criterias, especially if one is interested in the beginning and ending steps of the `design` phase.
  – The *exclusion* field indicates if direct criterias or sub-families are exclusive ($exc = e$) or not ($exc = ne$). For instance, there is in the repository a criteria related to project time pressure. Guidelines may be given for project under high time pressure as well as project under low time pressure. Therefore `time pressure` is a *family* and `low time pressure` and `high time pressure` are criterias. We specify them as exclusive criterias because guidelines associated to `high time pressure` project are not compatible with guidelines associated with `low time pressure` project and could not be provided in the same *solution* to a ISD *problem*.

  The *exclusion* field is associated to the family because if only some of the sub-nodes are exclusive among them, it means that a sub-family has to be isolated for these specific sub-nodes.

**Definition 1 (Context frame).** A context frame, $CF$, is a tree where:

- the root node is defined as <name=*base*, exc=*ne*, ord=*no*, type=*root*>,
- non-leaf nodes are *family* defined as <name, exc, ord, type=*family*>,
- leaf nodes are *criterias* defined as <name, exc, ord, type=*criteria*>

Figure 3 shows the structure of the *context frame*.

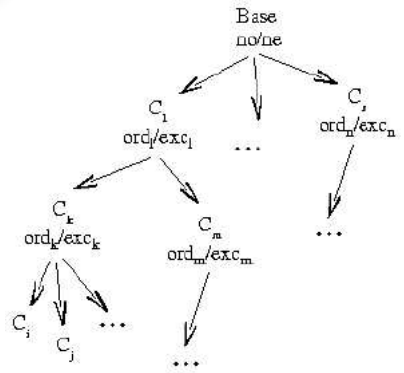


Figure 3: Context frame

**Definition 2 (Criteria).** A criteria is fully defined as a path from the root node `base` to a node $n_n$ of the context frame.

$Cr = [base, n_1, ..., n_n]$ *with* $base, n_1, ..., n_n \in CF$

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

$$if\ type_{n_n} = family,\ exc_{n_j} \neq e$$

**Definition 3 (Compatibility).** Two criterias are compatible if they do not share in their definition a common *family* node $n_i$ with an *exclusion* field equal to $e$.

$$Cr_1\ comp\ Cr_2$$
$$\forall n_i \in Cr_1\ and\ n_j \in Cr_2$$
$$if\ n_i = n_j\ then\ exc_{n_i} \neq e$$

**Definition 4 (Context).** A context is defined as a set of *compatible* criterias.

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

An example of *context frame* is given in figure 4. Three families of criterias are presented in this figure: `Designer expertise`, `Software to develop` and `Project`. The `Designer expertise` family helps in specifying for whom the guidelines are dedicated to: `Experts` are distinguished from `Medium` and `Beginners`. This family is specified as *ordered* because guidelines associated with the `Beginner` criteria may for instance complete guidelines associated with the `Medium` criteria when someone doesn't know exactly how to consider the expertise of the designers under consideration. And they are specified as *non-exclusive* because it could be meaningful to extract for instance fragment dedicated to `Medium` and `Beginner` designers in a same solution.

The `Software to develop` family groups criterias related to the qualification of the software to be developed (Mirbel and de Rivieres, 2002a). We distinguish the fact that the application to be developed includes a user-interface (`UI`), a database (`BD`), is distributed (`Distributed`) or is built on top of a running application (`Running software`). This last criteria is presented as a *family* because different aspects of a running software may be considered : `Functional domain`, `Interface` and `Code` (Mirbel and de Rivieres, 2003). Again a distinction is done among weak, medium and strong reuse of existing code, functional domain and interface. The `Software to develop` criteria is described as *non-ordered* because the different sub-families are non-related criterias. And they are described as *non-exclusive* because they may be associated with a same problem, solution or fragment. `Weak`, `Medium` and `Strong` criterias are considered as *non-ordered* because a fragment dedicated to guidelines to weakly keep the code for instance may not necessary be complementary to the one dedicated to keep the code in a medium way. And on the contrary, it is not forbidden also to search for a solution including for instance `Medium` and `Strong` criterias. Therefore, they are specified as *non-exclusive*.

The last family, `project`, groups criterias characterizing the project. It has been shown that it is important to take into account project features when working on methodological aspects of ISD (van Slooten and Hodes, 1996). In this example, `Delivery strategy`, `Realization strategy` and `Time pressure` aspects are taken into consideration. The three *families* are examples of *exclusive* criterias: there is only one `Delivery strategy` chosen for a project and the different kinds of strategies (`At once`, `incremental` and `evolutionary` ) can't be mixed. The remark is the same for the `Realization strategy`. With regards to `Time pressure`, the associated criterias are also *exclusive*, because guidelines for project under a `high time pressure` are not compatible with guidelines dedicated to project under a `low time pressure`.

An example of *criteria* taken from figure 4 is: `[base - software to develop - running software - code - weak]`. An example of *context* is {`[base - software to develop - running software - code - weak]`, `[base - software to develop - DB]`, `[base - Designer expertise - Medium]`}

# 5 CONTEXT FRAME IN USE

The *context frame* is a key element of our framework. In the following, we show how it is useful for the methodologists and for the project team members to support ISD *for* reuse and ISD *by* reuse.

Indeed, one unique context, called the *global context*, is created and maintained by the methodologists in the company. They are the only one allowed to build the *global context* customized for the company development process. It represents the ontology shared by all the projects and project team members with regards to ISD. The *global context* is used by the project team members when they build new fragments and when they search the repository for fragments organized in route-map dedicated to their problem. Contexts associated to fragments and problems, as it will be explained in the following, are subsets of the *global context*.

## 5.1 Context and criteria management

A *global context* is defined and managed by the methodologists. It allows them to define a single working space shared by all the projects and project team members in the company.

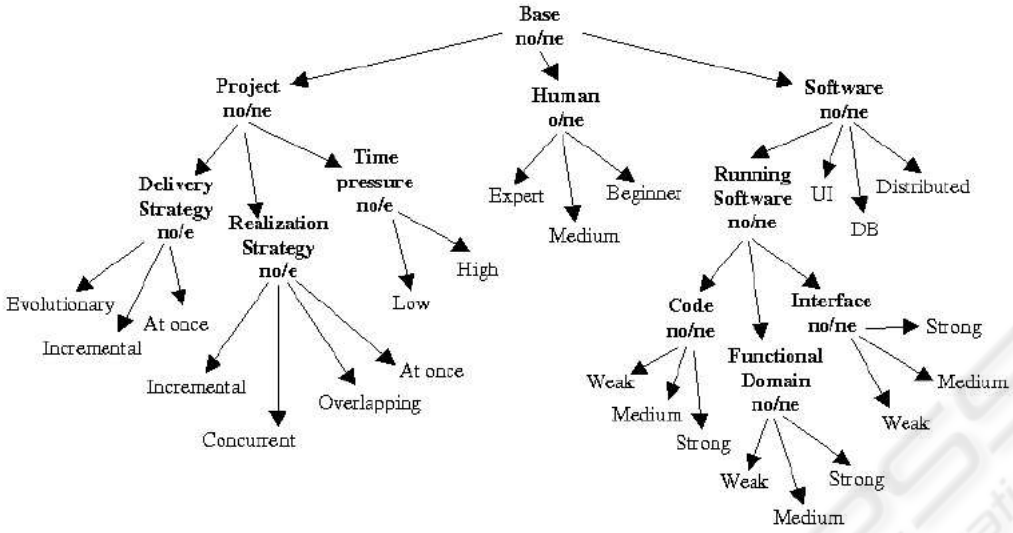


Figure 4: Example of context frame

**Definition 5 (Global context).** The *global context* is defined as a set of (at least one) compatible criterias.

$$GC = base, C_1, ..., C_n$$

The *global context* is shared by all the project team members who select in it the criterias they are interested in when building a new fragment or when searching for fragments in the repository. Project team members can not add new criterias. They have to use the ones defined by the methodologists. Methodologists make the *global context* evolve by refining existing criterias or adding new nodes under the *base* node.

Evolution is a key issue of the structure because it allows to always drive the project team members to focus on critical aspect(s) of the development whatever the process evolution is.

When they refine existing criterias, methodologists have to take care about the associated fragments. Indeed, fragments are always associated to *criteria* node of the global context. If a *criteria* node $n$ (leaf node) is refined into a *family* node (non-leaf node) with *criteria* sub-nodes $n_1, .., n_n$ (leaf nodes), then methodologists have to modify each fragment associated to $n$ in order to associate it to at least one of the sub-nodes $n_n$ ($n_n \in n_1, .., n_n$).

## 5.2 Context and ISD for reuse

A *fragment context* is associated with each fragment. It helps in situating the fragment in the repository with regards to the aspects of the development process which have been emphasized by methodologists; it anticipates fragment reuse and allows to provide a suitable solution to a given ISD problem when searching the repository.

**Definition 6 (Fragment context).** The *fragment context* is defined as a set of at least one compatible criteria taken from the *global context*.

$$FC = \{C_1, ..., C_n\}$$
$$\forall C_i \in FC, C_i \in GC$$
$$\forall C_i, C_j \in FC, C_i \; comp \; C_j$$

Each criteria constituting the *fragment context* has to end with a *criteria* node (leaf node):

- by definition it is not allowed to end a criteria with a *family* node (non-leaf node) which *exclusion* field is equal to *e* (cf section 4).
- An *exclusion* field equals to *ne* would be equivalent to a *fragment context* including all the sub-nodes of the *family* node (non-leaf node) under consideration, which means that the criteria would not be discriminant.

Therefore, only *criteria* nodes (leaf node) are allowed as terminal nodes in the *criteria* definition of the *fragment context*.

$$\forall C_i \in FC, C_i = [base, ..., n_j], type_{n_j} \neq family$$

## 5.3 Context and ISD by reuse

A *problem context* is associated to each problem (and its solution expressed in terms of fragment route-maps) to help in focusing on solutions dedicated to the problem. By comparing the *fragment contexts* and the *problem context*, fragments are selected and adapted

solutions are elaborated. The *problem context* is given within the problem definition to search the repository. *Necessary* criterias indicate aspects the project team member is interested in. *Forbidden* criterias indicate aspects the project team member is not interested in. It could be useful to specify it sometimes to be sure the fragments including these (forbidden) aspects will be removed from the solution before it is presented.

**Definition 7 (Problem context).** A *problem context* is defined as a set of at least one compatible *necessary criterias* and a set of compatible *forbidden criterias*. All criterias are taken from the *global context*.

$$PC =< CN, CF > \text{where}$$
$$CN = \{C_1, .., C_n\}, CF = \{C_1, .., C_m\}$$
$$\forall C_i \in CN, \nexists C_i \in CF$$
$$\forall C_i \in CN, C_i \in GC$$
$$\forall C_i \in CF, C_i \in GC$$
$$\forall C_i, C_j \in CN, C_i \; comp \; C_j$$
$$\forall C_i, C_j \in CF, C_i \; comp \; C_j$$

By definition, it is not allowed to end a criteria with a *family* node (non-leaf node) which *exclusion* field is equal to *e* (cf section 4). And an *exclusion* field of the terminal node describing the criteria equals to *ne* is equivalent to a *problem context* including all the sub-nodes.

# 6 CONCLUSION

In this paper we presented the *context frame*, a polymorphic structure to help supporting evolvability and scalability of ISD processes through knowledge capitalization and sharing. The framework we propose is mainly based on a *process fragment repository* and a *global context frame*. The *process fragment repository* is dedicated to the storing of process fragments which may be reused through ISD *by* reuse. The *global context frame* supports evolvability of ISD processes through the use of criterias to better specify *fragments*, *problems* and *solutions*.

From the methodologists point of view, there is a need for a common framework for all the project team members working in the company and for means to keep project team members in the boundaries of the company development process. The *context frame* encourages project team members to focus on specific/critical aspects of the project they are involved in and the development process they use. It should help project team members to always take as much advantage as possible from the last version of the development process chosen, adapted and used in the company. It is a scalable structure which supports evolution and tailoring by the methodologists for the project team member's need with regards to project and process features.

From the project team member point of view, means are provided:

- to help to select the right fragments to solve his/her problems and
- to allow him/her to qualify its reusable element of solution when he/she add it as a new fragment in the repository.

The *context frame* we propose answers these needs. Criterias are a means to closely match project team member's need and to take into account their evolution.

A case tool is under development to validate the approach on a real case in companies where attempts have already been made to customize the development processes and to provide dedicated solutions through process fragments (Mirbel and de Rivieres, 2002a).

In the future, we would like to weight fragments with regards to the expertise level of the project team members introducing the fragments into the repository. We will also introduce support to fragment comparison needed when entering a new fragment in the repository.

As the main goal of this approach is still to benefit from the experiences acquired during the resolution of previous problems, it is also crucial for us to provide means to capitalize information about the way fragments and route-maps are reused through the proposed framework. Therefore, our future works include the integration of *tracking information* to capitalize about the way ISD *by* reuse is handled.

# REFERENCES

Boehm, B. (1988). A spiral model of software development and enhancement. *Computer*, 21:61–72.

Brinkkemper, S., Saeki, M., and Harmsen, F. (1998). Assembly techniques for method engineering. In *10th International Conference on Advanced Information Systems Engineering*, Pisa, Italy.

Cauvet, C., Rieu, D., Fron-Conte, A., and Ramadour, P. (2001). *Ingnierie des systmes d'information*, chapter Rutilisation dans l'ingnierie des systmes d'information, pages 115–147. Hermes.

Finkelstein, A., Kramer, J., and Goedicke, M. (1990). Viewpoint oriented software developement. In *Le gnie logiciel et ses applications*, Toulouse, France.

Fowler, M. (1997). *Analysis Patterns: Reusable Object Models*. Object Technology Series. Addison-Wesley, Reading, Massachusetts.

Franckson, M. and Peugeot, C. (1991). Spcification of the object and process modeling language ESF. Technical Report D122-OPML-1.0.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY.

Gnatz, M., Marschall, F., Popp, G., Rausch, A., and Schwerin, W. (2001). Modular process patterns supporting an evolutionary software development process. *Lecture Notes in Computer Science*, 2188.

Henderson-Sellers, B. and Edwards, J. (1990). The object-oriented systems life cycle. *Communications of the ACM*, 33(9):142–159.

Mirbel, I. and de Rivieres, V. (2002a). Adapting Analysis and Design to Software Context: the JECKO Approach. In *8th International Conference on Object-Oriented Information S ystems*.

Mirbel, I. and de Rivieres, V. (2002b). Introducing Flexibility in the Heart of Analysis and Design. In *6th world multiconference on systemics, cybernetics and informatics (SCI)*.

Mirbel, I. and de Rivieres, V. (2003). *UML and the Unified Process*, chapter Towards a UML profile for building on top of running software. IRM Press.

Object Management Group (2001). Uml specification.

Potts, C. (1989). A generic model for representing design methods. In *11th International Conference on Software Engineering*.

Ralyte, J. (2001). *Ingenierie des methodes a base de composants*. PhD thesis, Universite Paris I - Sorbonne.

Rolland, C. and Souveyet, C. (1995). An approach for defining ways-of-working. *Information Systems Journal*.

Royce, W. (1970). Managing the development of large software systems: Concepts and techniques. In *WESCON*.

Saeki, M. (2003). Toward automated method engineering: Supporting method assembly in came. In *First International Workshop on Engineering methods to support information systems evolution*, Geneva, Switzerland.

si Said, S. and Rolland, C. (1998). Formalising guidance for the CREWS goal-scenario approach to requirements engineering. In *Eight European-Japanese Conference on Information Modelling and Knowledge Bases*.

van Slooten, K. and Hodes, B. (1996). Characterizing IS development projects. In S. Brinkkemper, K. Lyytinen, R. W., editor, *IFIP TC8, WG 8.1/8.2*, pages 29–44.