REUSABLE PROCESS CHUNKS*

Colette Rolland
Université de Paris Sorbonne
17, rue de la Sorbonne
75231 Paris Cedex 5
France

Naveen Prakash
Delhi Institute of Technology
Kashmere Gate, Old IG Block
110006 Delhi
India

Abstract. Reusability of project components, either at the code level or at the conceptual specification level, is considered a fundamental aspect in application development. More recently it as been argued that project histories can support reuse of design decisions. We propose a solution based on so-called process chunks which are generic process frames to resolve the issue stated by a generic requirements engineering (RE) situation. Chunks are classified into micro-chunks and macro-chunks. The latter support the decision making process whereas the former help in implementing the decision. The RE activity based on these chunks consists of (1) recognising a situation as belonging to the class of situations treated by a chunk and (2) the instanciation of the process frame. The benefit expected from this approach is threefold : (1) to speed up the requirements engineering process; (2) to improve the quality of the conceptual specifications; and (3) to offer an extensible way for capturing generic RE process knowledge.

1 Introduction

Reuse of project components is being put forward as a promising approach in the application development process [Biggerstaff1987], [ACM1990], [Gibbes1990] . In fact it facilitates both effective development of good quality applications by exploiting existing validated components, and easy re-development of applications by promoting a modular approach. It is commonly agreed that reuse does not only concern code, but any development artifact. Thus, requirements specifications, design decisions, logical specifications are also candidates for reuse. A more recent claim is that the development process itself is candidate for reuse (and not only the artifacts which result of the process). A common characteristic of recent proposals is the representation of the development history as a set of decisions which transform the state of artifacts [Jarke1988], [Jarke1992], [Potts1989], [Balasubramanian1992] and the argumentation for reusing design decision in subsequent projects.

In this paper we address the problem of reusing generic pieces of the Requirements Engineering Process which are called Process Chunks. Requirements Engineering is this early part of the system development which involves investigating the problems and requirements of the users community and developing the conceptual specification of the future system, the so-called conceptual schema. The Requirements Engineering process is the route to reach the conceptual schema. It has the purpose of abstracting and conceptualizing relevant parts of the application domain. This is guided by requirements [Dubois1987].

* This work has been partially supported by the ESPRIT project NATURE

A process chunck is a piece of generic knowledge reusable for requirements engineering issues of the same kind. It is in the form <situation, decision, argument, action> and associates to a common situation in requirements engineering, the decisions the requirements engineer can take in that situation, the arguments to choose among the possible decisions and the action which is a schema transformation that is triggered by the engineer choice. A process chunk is a knowledge store which is structured according to a process model. The process model underlying process chunks is presented in section 2. In section 3, the notion of process chunck is defined and examplified. The RE-by-reuse methodology is sketched in section 4. Our concluding remarks are given in section 5.

2 The process model

According to Dowson [Dowson1987] existing process models can be classified into three categories:
-*Activity-oriented models* [Royce 1970], [Boehm 1988], [Henderson-Sellers & Edwards 1990] come from an analogy with problem solving, i.e. finding and executing a plan of actions leading to the solution. They are sequential in nature and provide a frame for manual management of projects developed in a linear fashion. Such a linear view of the design process is inadequate for methodologies which support backtracking, reuse of previous designs and have to support parallel engineering activities.
-*The product-oriented process models* [Finkelstein & al. 1990], [Peugeot and Franckson 1991]. represent the development process through the evolution of the product. They promote a view of development processes which is still centred around the notion of development activity but emphasises the linkage between the development activities and their output, the product.
-A more recent class of process models follows a *decision-oriented paradigm*. The successive transformations of the product are looked upon as consequences of decisions. The process model of the DAIDA project [Jarke & al. 1990; Rose & al. 1991] and of [Potts 1989] fall into this category. Such models are semantically more powerful than the previous ones because they explain not only how the process proceeds but also why transformations happen. They provide a more complete knowledge about the process which is particularly useful for reuse of developed products, or of design decisions, and for backtracking purposes when a modification in requirements has to be considered either during the design process itself or during the maintenance of the information system

Considering the highly non-deterministic nature of the Requirements Engineering process we do believe that only the last of these approaches appears to be partially appropriate. It is probably difficult to impossible to write down a realistic state-transition diagram that adequately describes what has to happen or actually happens in requirements engineering. But relying on the pure artifact history is also insufficient. Even the decision-based approaches offer only limited hints about when and how to decide on what.

Analysts react *contextually* according to the domain knowledge they acquire and react by analogy with previous situations they have been involved in. In order to take into account the very nature of the RE we have chosen to emphazise the contextual aspect of decisions. Our process modeling approach strongly couples the context of a decision to the decision itself, otherwise we loose information about the decision. This approach aims at capturing not only activities performed during the RE process but also why these activities are performed (the decisions) and when (the decision contexts).

The central aspect of our process way of modeling is therefore that it makes the notion of *situation* (in which to decide) explicit and relates it to the broader question of context handling. Figure 1 gives an overview of the process model, introducing its key concepts and their relationships (with a binary E/R based notation).This approach is an extension of the one presented in [Grosz&Rolland1990] and implemented in the ALECSI prototype [Rolland&Cauvet1991].

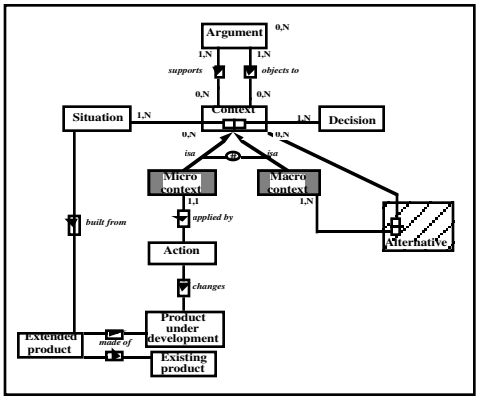


Fig.1 : Simplified overview of the process model

The four basic concepts are the ones of situation, decision, action and argument :
- *situation* to explain the decision context,
- *decision* to guide the RE process,
- *action* to perform the product transformations,
- *argument* to support decision-making.

A *situation* is most often a part of the product under development it makes sense to take a decision on. It represents the context of a decision. It can be a statement about the application domain or a requirement. It can also refer to part of existing products resulting from previous developments or a piece of generic domain knowledge. Accordingly, in figure1, the situation *is built on* either the *product under development* or *existing products*. A complete product model is presented in [Schmitt1993].
A *decision* reflects a choice that a requirement engineer makes at a given point in time of the RE process. A decision can be looked upon as the intention of product transformations. It explicitly represents the resolution of an issue.

An *action* performs a transformation on the product. It is a materialisation of the decision. Performing it changes the product and may generate new situations which, in turn, are subjects of new decisions.
Arguments are statements which support or object to decisions within a given context.

The model emphasises the dimension of decisions and introduces the notion of context as a main concept of process modelling .A *contex;*t is a couple <situation,decision> which amplifies the decision semantics by refining the "when" part of the decision.
Furthermore, the process model has to handle different levels of granularity of contexts, i.e. granularity of situations, as well as of decisions. This leads to the specialisation of the context into *macro* and *micro* ones. The former corresponds to a situation which requires to explore alternatives in decision making for its treatment. This is represented in Figure 1 by the loop upon context through macro context and alternative. This introduces the decomposition of a macro context into its more elementary context. Decomposition can be regarded as refinement in decision making. On the other hand, when the context cannot be associated to multiple ways of solving the issue it raises then it is referred to as a micro context. Such a context implements one decision through execution of actions which transform the product under development. This raises new situations which are themselves subjects of decisions. This is shown in Figure 1 by the loop upon context through micro context, action, and situation. In this model the granularity in decision making relates to consequences of decisions. Some decisions have consequences that are only product transformations whereas others imply the taking of decisions as consequences. The former are referred to as *micro decisions* and the latter as *macro decisions* ; A macro decision can be qualified as a strategic one whereas the micro decision implements a tactic.
Consequently, the requirement engineering process is decision oriented and recognises two kinds of decisions, the strategic and the tactical. The former needs to be refined into tactical decisions through a certain number of steps of decision making. A tactical decision can be immediately implemented through a set of actions.

3 The notion of process chunk

A process chunk captures generic process knowledge. It is context-targeted and intends to provide help to the requirements engineer for reacting to the RE situations he is faced to, through decision making and associated actions. A process chunk is concerned with one generic context composed of one generic situation that might occur in the RE process and an associated generic decision. According to the classification of contexts introduced earlier, chunks will be classified into macro and micro chunks.

The macro chunk corresponds to a situation which requires the exploration of alternatives in decision making. Each alternative is an approach or a strategy for the resolution of the issue being faced by the engineer in the current situation. The chunk supports the requirement engineer in the exploration of these strategies and in the selection of the most appropriate one. By definition, a macro chunk offers a choice

among a set of strategies all of which achieve the same purpose. In this sense, one can look upon the macro chunk as being goal oriented.

A micro chunk refers to a situation for which the strategy to be followed has been decided. It supports the implementation of the tactic associated with the strategy through the ordering of actions to be performed on the product under development.

3.1 The Macro Chunk

The structure of the macro chunk is given in Figure 2. As can be seen the main notion underlying a macro chunk is that of macro context. Basically, the macro chunk provides support to decompose the initial macro context into other contexts. This is done through deliberation on the produced contexts by arguing in favour of or against them. Thus the key concepts underlying the macro chunk are the ones of context, alternative, and argument.

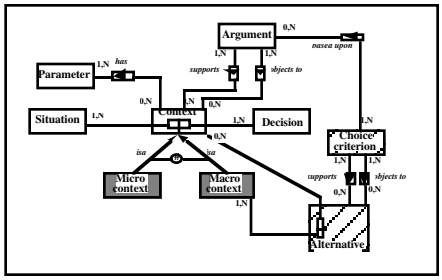


Fig 2 : Detailed view of a macro chunk

A macro chunk is associated to a macro context that shall be referred to as the mother context. This is composed of a generic pair, <situation, decision> and associated parameters. As an example, consider the mother context, MC, shown in Figure 3 consisting of

- the generic situation defined by the binary relationship between two entity-types,
- the generic decision, 'structure', and
- the parameters which characterise the relationship using cardinalities. Cardinalities are of three types, totality, valuation, and permanency. The first two of these are the usual ER ones and take on values as follows

totality : *partial* (p) or *total* (t), {0 or 1 respectively of the ER minimum cardinality }

valuation :*simple* (s) or *multiple* (m) {1 or N respectively of the ER maximum cardinality } The third one is specific and takes on the following values

permanency :*variable* (v) or *permanent* (p)

The relationship between E1 and E2 is said to be *permanent* if an instance of E2 which is associated with an instance of E1 remains permanently linked to it during the full life cycle of E1. The relationship is said to be *variable* when an instance of E1 is related to different instances of E2 during its life cycle.

As shown in Figure 3, there are two triplets of cardinalities which characterise respectively the relationship from E1 to E2 and the one from E2 to E1.

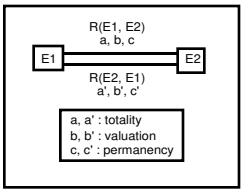


Fig 3: Example of situation in a mother macro-context,

The macro chunk aims at refining the decision of the mother context by considering various alternatives all of which are relevant to the resolution of this decision. This is modelled in terms of the decomposition of the mother context into other contexts. For instance a partial list of contexts resulting from the decomposition of MC is as follows:

- < {E1, R(E1,E2)}, 'partition'> (define two disjoint sub-types)
- < {E2,R(E2,E1)}, 'partition'>
- < {E1, E2, R(E1,E2)}, 'embed'> (define a composite object by embedding E2 into E1)
- < {R(E1,E2)}, 'retype'> (mutate the relationship into an entity type)
- < {R(E1,E2)}, 'transform'> (transform the relationship into an ISA relationship)
- < { E1, R(E1,E2)}, historise> (keep track in E1 of the evolution of its reltionship with E2)
- < { E2, R(E2,E1)}, historise> (keep track in E2 of the evolution of its relationship with E1)

In order to help in the selection of the appropriate alternative, the chunk provides criteria for choosing among these (hatched box in Figure 2). A choice criterion combines arguments which support/object to the decision within a context. Taken together, the choice criterion and arguments make explicit the deliberation behind the decision making process. Along the lines of the IBIS model [Conklin&Begeman1992] the chunk helps the requirement engineer in acting according to the rationale of decisions. In the example considered above, the arguments will be based on values of cardinalities. For example the cardinalities triplet

< p, -, - > where '-' could take on any of the allowed value

associated with the relationship R(E1,E2) in the forward direction supports the partition of E1. Similarly, when associated with the relationship in the backward direction, this triplet supports the partition of E2.

It is possible to base the arguments associated with 'embed' in a similar manner. The two cardinality sextuplets

< < t,m,p> <t,s,p > > and < <t,m,v> <t,s,p,> >

associated with the relationship between E1 and E2 support the embedding of E2 into E1. Due to the valuation, m, there shall be multiple instances of E2 embedded in one instance of E1. Similarly, the two cardinality sextuplets

< < t,s,p> <t,s,p > > and < <t,s,v> <t,s,p,> >

associated with the relationship between E1 and E2 support the embedding of E2 into E1. Due to the valuation, s, there shall be one instance of E2 embedded in one instance of E1.

The choice criterion is built up using arguments. For instance, a criterion could be formulated as follows

< if the argument supporting specialisation of an entity type holds then this decision has a higher priority over any other decision supported by another argument >. One can therefore notice that a choice criterion has to be formulated when several arguments hold at the same time. The choice criterion gives, in fact, a priority to one of the arguments.

3.2 The Micro Chunk

A micro chunk is associated with a micro context, i.e., to a pair <situation, decision> which implies only one way of solving the issue it raises. One can say that the micro chunk is only concerned with the manner in which the decision shall be implemented and not with exploring alternatives. The implementation of the decision of the micro context will be performed by actions that transform the product under development. There are two major differences between the macro chunk and the micro chunk : the first one lies in the absence of any alternatives in the latter and the second one is that a macro chunk has no direct consequence on the product under development.

The structure of a micro-chunck is shown in Figure 4. As can be seen the main concept underlying the micro-chunk is that of micro-context. The micro-chunk provides the procedure to perform the transformations of the product under development resulting of the decision stated by the initial micro-context. This is merely achieved through the execution of actions but may require the decomposition of micro-contexts into more atomic micro-contexts. Thus the key notions of a micro-chunck are those of micro-context and action.

A micro-chunk is associated to a micro-context that will be referred to as the mother-context. Each of the contexts resulting of the decomposition of the macro-context < {E1, R(E1,E2), E2}, 'structure' > introduced in the previous paragraph are examples of such micro-contexts. A context consists of a decision and the situation subject to this decision. For instance the context < {E1,R(E1,E2)}, 'Partition' > is based upon a situation composed of an Entity-type E1 and the decision to partition E1.

If the consequence of the decision stated in the micro-context consists of a unique action, it is referred to as an *atomic context*. An *atomic context* triggers only one action which can be in turn (cross-hatched boxes in figure 4) *compound* (composed of several actions) or *atomic* (changing only one product element). In this case, the decision is directly implemented. Even though a commitment to implementing the decision has been made, it is possible that the implementation is complex requiring transformations of several elements of the product under development. In this case the context is said a *compound context.* A compound context is itself a pair <

situation, decision> that has to be further decomposed into more atomic contexts. This decomposition proceeds till atomic contexts are reached, which, by definition, are immediatly implementable. This is modelled through the decomposition of contexts ("composed of" relationship in figure 4).

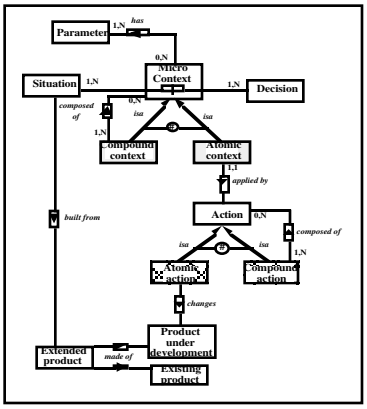
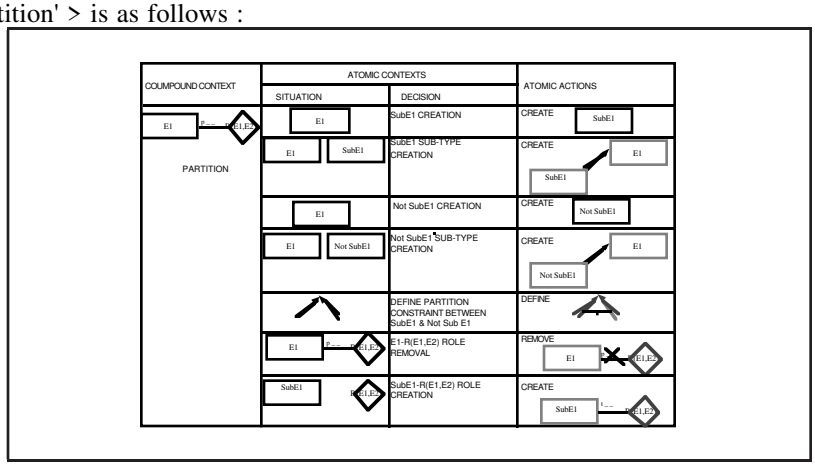


Fig 4 : Detailed view of the micro chunk

For instance, the decomposition of the compound context < {E1, R(E1,E2)} ,'partition' > is as follows :



The mother micro-context <E1, R(E1,E2), 'partition'> is further decomposed into seven atomic contexts including each, a sub-micro situation of the initial one and the corresponding micro-decision. Each micro decision is implemented through the execution of one action which creates or removes elements of the product under development. For instance the first atomi-type E1 and the micro-decision 'SubE1 Creation' which results in the creation of a sub-type of the Entity-type E1 . The fith atomic context associates to the two sub-typing links introduced to the Entity-type E1, the decision of defining a partition constraint which is immediatly implemented by the correspondig action.

5 Reusing process chunks

In this section we address the problem of reusing process chunks. Reusability can be considered from two different perspectives. According to a *design-by-reuse* perspective an application can be obtained by tailoring existing components extracted from an available repository. According to a *design-for-reuse* perspective, a repository must be populated with reusable components, properly defined to be exploitable for developing new applications not from scratch. We sketch in this section a RE-by-reuse approach in which the Requirements Engineering process is guided by the reuse of process chunks. Process knowledge is captured into a Process Knowledge Base wich consists of a collection of process chunks.

From an operational point of view, the Chunk definition is aimed at providing guidance for method engineers It helps these engineers in defining the way-of-working of the specific methodology they have to develop. The way-of-working of a given methodology will be obtained by instanciating macro and micro chunks as defined in the previous section The result of such an instanciation consists of a collection of chunks such as the macro chunk <{ E1, R(E1,E2), E2}, 'structure'> and the micro-chunk <{ E1, R(E1,E2) }, 'partition' > previously .discussed They will be stored in the Process Knowledge Base.

Once fully defined, the Process Knowledge Base can be used to *guide* the Requirements Engineering. Guidance will be provided by the Process Assistant by performing pattern matching. Indeed, the chunks of the process knowledge base can be matched against the current state of the product under development and help the developer in considering relevant situations and in taking the right decisions. Selection of a macro-chunk will help the requirements engineer in developing a top-down decision making.process This will allow him to select the rigth strategy to solve the issue he is faced to during the RE process When the initial macro-context has been further decomposed into micro-contexts, the corresponding micro-chunks will support the engineer in the tactical implementation of the chosen decisions.

Some experiments of this approach have been made through the development of the design expert system ALECSI [Cauvet&Rolland1991], [Rolland&Proix 1992] We examplified it with part of an Object-Oriented schema construction.
Let assume that the data schema under development is currently composed of the relationship R(Client, Product) as stated in Figure 5.1. The cardinality triplet is < *p, m ,v* > in the forward direction : a client may not have associated product(p), may have ordered several products (m) and is not always associated with the same products (v). The cardinality triplet in the backward direction is < *p, m, v* >*:* a product is not necessarily associated with a client (p), it may be linked to a number of clients (m) which vary in time (v).

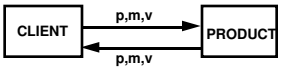


Fig.5.1 Example of situation

The macro-chunk MC1 : < {E1, R(E1,E2), E2}, 'structure' > presented in section III holds in this case .The situation considered in Figure 5.1 belongs to the class of situations of the mother context of MC1. According to the parameter values of the context <*p,m,v*> <*p,m,v*>, the macro-chunk suggests two possible kinds of decisions : either to partition the Entity-type Client as well as the Entity-type Product, or to refine the schema by retyping the Relationship-type into an Entity-type. Let assume that the decision chosen by the engineer is to refine the schema by retyping the relationship into the Entity-type Order. This corresponds to the decomposition of the mother context into the micro-context < {R(Client, Product)}, 'mutate' > which is instance of the micro-chunk <{R(E1,E2)}, 'retype' >providing the procedure to implement the 'retype' decision. This leads to the transformation of the schema as simply graphically showned in Figure 5.2 Some of the cardinalities are automatically infered, some have to be introduced by the engineer.

Fig. 5.2 Mutation of the relationship

Again the two relationships R(Client, Order) and R(Order, Product).match the situation of the macro-chunk MC1 and can be structured accordingly the advices provided by the chunk. Due to the sextuplet of cardinalities of the former relationship, the macro-chunk suggests three alternatives : partitioning Entity-types, historising the relationship-type and transforming it into a reference link between two object classes. Let suppose that the engineer chooses the last one. Figure 5.3 shows the resulting schema.

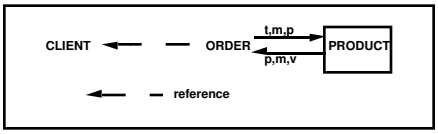


Fig. 5.3 Creation of object classes and a referencing link

According to the cardinalities of the relationship R(Order,Product), the possible decisions are : 'historise', 'referencing' and 'retype'. Figure 5.4 shows that the requirements engineer has found relevant to retype the relationship-type into the Entity-type Order-Line.

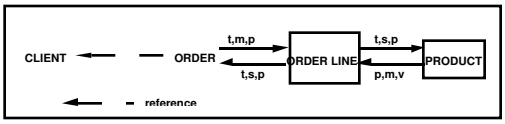


Fig. 5.4 Introduction of the Entity-Type Order-Line

The relationship R(Order, Order-Line) matches the situation of the macro-chunk MC1 and is transformed as presented in Figure 5.5 by embedding the order-lines into orders. Order-line becomes a component object class embendded into the composite object class Order.

Finally according to the cardinalities of the relationship R(Order-line, Product) the macro-chunk MC1 suggests the following decisions : 'historise', 'retype', 'partition' Product or objectify Product with 'referencing' Product from Order-line. The latter is chosen and the resulting schema is presented in Figure 5.5.

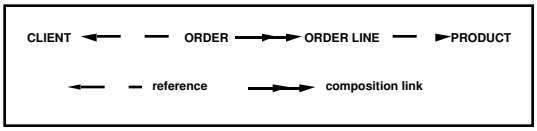


Fig. 7.5 Final schema transformation

As illustrated in this example, macro-chunks guide the RE process by matching situations which appear in the product under development with generic situations of mother contexts. The RE-by-reuse approach consists then, of suggesting possible decisions to the RE engineer and performing the consequences of the selected decision following the procedure stored in the corresponding micro-chunk.

5 Conclusion

In this paper, we presented an innovative and operational approach to build conceptual specifications. It is based on the reuse of RE process chunks. A chunk is a parametrized frame representing the possible ways to solve the issue raised in a common situation of requirements engineering. A chunk is expressed in a way which facilitates its reuse in subsequent projects.

The use of chunks changes the requirements engineering process. Building conceptual specifications is no longer a tedious composition of elementary concepts (e.g entity, event, operation, etc.), but rather the recognition of a particular situation as belonging to a chunk context and the instanciation of the process frame corresponding to the chunk.

In this paper we presented chunks in an informal way; However, it is clear that they can be expressed in different formalisms. As a matter of fact , we expressed chunks using the TELOS language and we are currently implementing the Process Assistant using the Knowledge Base System "Concept Base" [Jarke&al1992]

Our work goes on the same line as the ones defined for instance by Fugini [Fugini&al1991] in the Ithaca project or Pietro-Diaz [Pietro-Diaz1991] for domain analysis. With chunks, we are definitely "*identifying and structuring information for reusability"* . However, with chunks we are focusing on reuse of process patterns and not on reuse of domain knowledge or conceptual specifications.

We are currently working on the extension of the existing library to fully support the NATURE [Jarke&al1992] way-of working. We are also studying how process chunks could be discovered in a semi-automatic way using a learning by example tool.

6 References

[ACM1990] : ACM Issue on Object-Oriented Design, 1990

[Balasubramaniam1992] : Balasubramaniam. R., Vasant. Dhar " Supporting Systems Development by Capturing Deliberations During Requirements Engineering", IEEE Transactions of Software Engineering, Vol 18, No 6, 1992.

[[Biggerstaff1987] : Biggerstaff.T et al, " Information Management Challenges in the Software Design Process", IEEE Database Engineering, Vol 10, 1987.
[Boehm1988] : Boehm B.W. : "A Spiral Model of Software Development and Enhancement"; IEEE Computer 21.,1988.
[[Conklin&Begeman1992] : Conklin. J., Begeman. M. " gIBIS : a hypertext tool for exploratory policy discussion ", ACM Trans. on Office Information System, Vol 6, 1988.
[Dubois1989] : Dubois E.,Hagelstein J, Rifaut A. : "Formal Requirements Engineering with ERAE", Philips Journal Research, Vol 43, No 4, 1989.
[Dowson1987] : Dowson M. : "Iteration in the Software Process. Proc 9th Int Conf on Software Engineering", Monterey, CA,1988.
[Finkelstein et al. 1990] : Finkelstein A., Kramer J., Goedicke M.; "ViewPoint Oriented Software Development"; Proc. Conf "Le Génie Logiciel et ses Applications", Toulouse, p 337-351, 1990.
[[Gibbes1990] : Gibbes S, Tsichritsis.I, Casais.E, Nierstrazs. O, Pintado.X : "Class Management for Software Communities", Com. ACM, Vol 33, No 9, 1990.
[Grosz&Rolland1990] : Grosz G., Rolland C. : " Using Artificial Intelligence Techniques to Formalize the Information System Design Process"; Proc Int Conf on Databases and Expert Systems Applications, 1990.
[Henderson-Sellers and Edwards 1990] : Henderson-Sellers B., Edwards J.M.; "The Object-oriented Systems Life-Cycle"; Comm. ACM, 09, 1990.
[Jarke1988] : Jarke. M., Rose. T.," Managing Knowledge about Information System Evolution", Proc. ACM SIGMOD, 1988.
[Jarke et al1992] : Jarke M., Bubenko J, Rolland C., Sutcliffe A., Vassiliou Y.; "NATURE Initial Integration Report", NATURE.D-I, 1992.
[Jarke et al.1992] : Jarke M., Mylopoulos J., Schmidt J.W., Vassiliou Y.; "DAIDA : An environment for evolving information systems", ACM Transactions on Information Systems, Vol. 10, No. 1, 1992.
[Peugeot and Franckson 1991] : Peugeot C., Franckson M.; "Specification of the Object and Process Modelling Language", ESF Report n° D122-OPML-1.0, 1991.
[Potts 1989] : Potts C.; “A Generic Model for Representing Design Methods”; Proc. 11th International Conference on Software Engineering, 1989.
[Rolland&Cauvet1991] : Rolland C., Cauvet C. : "ALECSI : An Expert System for Requirements Engineering", in "Advanced Information Systems Engineering", R.Andersen, J.A.Bubenko, A.Solvberg (Eds), Springer Verlag, 1991.
[Rolland&Proix1992] : Rolland C, Proix C. : "A Natural Language Approach for Requirements Engineering", in "Advanced Information Systems Engineering", P. Loucopoulos, K; Jeffery, J.A Bubenko, A.Solvberg (Eds), Springer Verlag, 1992.
[Rose et al1991] : Rose T., Jarke M., Gocek M., Maltzahn C., Nissen H. : " A Decision-Based Configuration Process Environment", Software Engineering Journal, 6, 5.
[Royce1970] : Royce W.W.; "Managing the development of large Software Systems"; Proc. IEEE WESCON 08/1970.
[Schmitt1993] : Schmitt J.R., " Product Modelling for Requirement Engineering Process Modelling", IFIP Int. Conf. on " Information System Development Process", Rolland C., Prakash N., Pernici B. (eds), North Holland (pub), 1993.