Process centered SEE and Adele [*]

Jacky Estublier Noureddine Belkhatir Mohamed A. Nacer Walcelio L. Melo [†]

L.G.I. BP 53X, 38041 Grenoble Cedex, France
{belkhatir,jacky,nacer,wmelo}@hoggar.imag.fr

Abstract

This paper presents the role and the evolution of Software Engineering Environment(SEE) and Computer Aided Software Engineering Environments (CASE Environments) in large software development and maintenance. We emphasize the drawbacks of the current state of the art, advocating improved structural and behavioral modeling and the introduction of team support to meet software engineering requirements: what we need is a Process centered SEE its trigger mechanism with its data model to solve some team support problems.

Key-words: CASE; Software engineering environments, process modeling; programming in the large; active software database; product modeling;

1 Introduction

The development and maintenance of large systems is a complex task, where versioned documents of different types (design, program, test, etc.), actors (playing several roles), tools, methods, and policies must be managed and parallel activities must be coordinated. In this soft of context, everything changes, not only the managed objects (versions), but also the users, tools, methods and policies. The problem is to deal with this complexity in order to improve the quality of the produced system, the efficiency of the teams involved and the predictability of the schedule. Many **C**omputer **A**ided **S**oftware **E**ngineering environments (CASE) have been built to solve these problems; but they are not fully satisfactory. In order to support programming in the large activities, a CASE environment needs to be able to deal with two main kinds of information.The static information models software components: design, documentation, set of tests, programs, etc, their relationships, and their evolution versions. Dynamic information (behavior model, activity control) describes how to develop and maintain the product, share and synchronize the activities among users, integrate foreign tools, control tool execution, and finally how to describe and enforce policies (methods, procedures, conventions, etc). This paper presents a synthesis of CASE environments showing the different evolution for modeling information. The different approaches are described with emphasis on the different issues proposed to solve integration problems as well as the evolution of the techniques for software engineering. Finally we present the Adele environment as an example of customizable CASE environment.

2 CASE Environment Evolution:

CASE environments have evolved during the last 20 years. Among the available CASEs, we can distinguish three major evolution steps: The file System step (where the systems were built from scratch using standard File Management Systems); the database step (where systems were built upon classical DBMS); and the CASE step where systems were ad hoc or built from specialized kernels with custom-built CASE tools.

[*]Published in the *Proc. of the 5th Int'l Workshop on Computer-Aided Software Engineering (CASE'92)*, Montreal, Quebec, Canada, July 6–10, 1992. IEEE Press

[†]Melo is supported by Technological and Scientific Development National Council of Brazil (CNPq).

2.1 File system approach

These systems were the first to be built, and also the first to tackle programming-in-the-large problems. They progressed at least in three directions:

- Version management. Sccs defined the concept of revision, and solved the disk space problem (the delta mechanism), the history problem and the multi-user problem (freezing mechanism).
- Tool control and rebuilding. The Make tool represented a breakthrough in 1976 and this popular tool is still much used.
- System structure. The concept of System Model, proposed by Gandalf influenced all the following SEEs. Environments like Cedar [Swi86], Dsee [Leb84], Nse [Sun88], and Jasmine [Mar86] use this concept. It defines the structure of large software product in terms of components interconnected by "dependency" relationships, describing the relations between components, the information on the versions and the rules of software building. Reasoning on this "dependency" information leads to a better understanding of software structure.

These systems use a standard File Management System (Unix, Sccs, Rcs) or a slightly extended one (Gandalf) as a repository. As a consequence, little is known about object characteristics and its interconnections.

A significatn drawback with this approach is that little was done to take account of dynamic aspects. In line with these aspects, two sub-generations can be distinguished: the toolkit generation where the only mechanism provided for activity coordination was the operating system language; and the hardwired generation where predefined and inflexible policies are used to enforce coordination and team communication, and to control shared resources e.g. Dsee, Infuse and Nse.

2.2 Data Base approach

The concepts developed so far, such as the system model, are easily modeled by objects and relationships. Since classical databases provide such concepts, using a data base as the basic platform seems to be a logical approach.

Unfortunately, most work in this direction failed because of a mismatch between the DB technology (relational) available at that time (early 80s) and SEE requirements. The experimentations conducted on the relational DBMS showed that they are not adapted to support SEEs [Pen91, Ber87, Unl89], because they are developed mainly for commercial applications. As a result, these systems were not satisfactory for example when managing files (long fields), revisions, system rebuild, schema evolution, and distributed work (work space control).

2.3 The current CASE

CASE environments tried to provide a wider scope, more life cycle phases and better support for workers by making the software process explicit. This evolution led to two results:

- the kernel oriented CASE where the emphasis is on a specialized kernel with independent tools around it (Pcte[Bou88], Cais[Obe88], Adele1, Atis). Such CASEs do not provide enough facilities for software processes management.
- the monolithic CASE tool which integrates object management and predefined policies along with a predefined set of tools. Almost all currently used CASEs are in this group. They use a commercial database as support (Andromede), an ad hoc database, or even a standard file management system. The emphasis of these systems can be one or more of the following topics: version control (Dsee, Rcs), long transactions (Nse), configuration building (Adele1, Aide-de-camp).
- The PSEE, Process centered SEE, that integrate the static aspects (data and product modeling) and the dynamic aspect (process modeling and enacting). This integration significantly increases power and represents a new approach to software development paradigms. Adele 2, SoftBench, Arcadia, Epos, Alf and Marvel are the early results of this new approach.

It looks as if there is a consensus on the capabilities a PSEE must provide, namely:

- A versioned repository where all software artefacts can be stored and fetched. A repository must be driven by a data model incorporating mainly ERA and OO concepts. The first one is better adapted to modeling software structures, and the second one to modeling the behavioral aspects of the SEEs.
- Executable formalisms for the explicit description of the activities and their decomposition.
- A Long Transaction facility, in order to implement Work Space control and sub_database.

Projects which try to integrate aspects of products and tasks into a single object model are for instance Epos, Alf/Pcte and Adele .

Rather than modify the OMS of Pcte, Alf [Zuc91] extends it to process aspects. This development environment is based on the knowledge of the plan of the software development and of various existing projects.

Epos is a CASE prototype built around an object-oriented database, a configuration manager, and a process manager, all sharing the same data model: an extension of the ERA model to the object oriented concepts.

The Adele data model (ERA) has been extended to support Object Oriented models and complex object management. An Adele 2 prototype has been developed which integrates those aspects and mechanisms for process management [Bel91a, Bel91b]. In the section 4, we present Adele 2 in greater detail showing how it can be used as a customizable PSEE.

3 The technology evolution

CASE environment evolution can also be considered from the point of view of the evolution of techniques. The major technologies involved are:

- The Database technology,
- The Software Engineering technology,
- The Process Modeling and AI technology.

3.1 The Database towards Software Engineering

Many projects were interested in the database technology because of their modeling power and commercial availability. In other words, objects and relationships are explicitly modeled. Experience has clarified the following drawbacks of these projects:

- it is difficult to model complex objects. Relational Databases have an unstructured view of the objects.
- Lack of encapsulation. The operations applicable to an object are not identified, their semantic are left completely to the applications.
- In general, a lack of mechanisms for dealing with dynamic aspects (events, constraints, triggers, methods, etc) making it possible to manage complex applications.

Most authors agree that currently available databases do not provide the following SEE requirements: (1) most objects have a name, they are complex (aggregate management); (2) objects are evolving (version management on heterogeneous objects); (3) users, tools, methods, and policies are evolving (schema evolution); (4) consistency (or rather inconsistency) management (complex relationships); (5) structuring of teams, products, and work environments (types and instances); (6) powerful viewing mechanisms (encompassing active paradigms); (7) broad scope of knowledge (tools, users, methods, objects, etc.); (8) deep interconnection between Configuration Management, Data and Process as (CM/PM); (9) transactional model (long transactions).

Databases evolved mainly in two directions: the Object Oriented DBs O2 [Lec89], Orion [Kim88], integrating OO programming languages with DB, and the semantic models extending the ERA paradigm Damokles [Dit89] more closely connected with software engineering.

3.2 Software Engineering towards the Database.

Software engineers designed databases for their own purposes. Logically these DBs can be seen as File System extensions to integrate Entity Relationship concepts. Pcte, Cais, Atis and Adele are platforms of this type. The basic idea is that a file system is conceptually too low and its modeling power not sufficient, and that tool integration can only be effective when sharing a common data schema.

This evolution led to a sharp boundary between the platform (a passive object server) and the tools and activities that must be built on top of it. No real services are provided for activity control (this is why projects such as ALF are under way), not for the tool control.

From another point of view, the modeling power of these platforms is not that high; no Object Oriented concept is provided and difficulties remain at least in versioning, complex object management, and team support. Both database and software engineering results led to a new kernel generation. The solutions considered in most kernel development projects try to make a compromise between recent database technology and the integration of mechanisms proper to programming environments.

In the current trends, DB for software engineering integrate both approaches : ERA DB extended with OO features with better support for versioning [Zdo86], structuring and consistency control. Products like PCTE+ [Gal86] and Adele 2 [Est92] are of that class. The PSEE approach usually starts from such a platform and adds process modeling.

3.3 Process modeling towards Software Engineering

Over the last years, studies have concluded that better control of the processes involved in development and maintenance is a necessary condition for obtaining better products [Dow91, Tho91]. We lack methods and tools for developing large products and managing their evolution. In other words, we need to control software processes.

In order to introduce some degree of discipline into the process, policies must be formalized and enforced to control, monitor and assist teams to perform their activities. A process model must provide mechanisms to:

- describe the different activities carried out during the software process;
- control the ordering, synchronization and concurrency among activities;
- model the objects produced and consumed by activities;
- integrate tools used by the activities;
- define and control users involved in the activities.

Process modeling is a fashionable topic and, despite the large number of current projects (more than 30 all around the world), little practical results have been arrived so far. However significant progress has been made over the last two years and practical results will be available soon.

Several prototype systems have been built to support these dynamic aspects, but for the time being, no consensus has been reached on the paradigms and approaches to use. It seems that no single approach can solve all the problems. The current paradigms are the following:

3.3.1 Rule-based paradigm.

In this approach, the knowledge about the activities and tasks of a generic software development process is explicitly modeled by rules using AI techniques are used (planning, blackboards). The tools are integrated into the environment through the utilization of pre and postcondition over their inputs and outputs. The rules may be different depending on the implementation chosen by the system (backward and/or forward reasoning, static or dynamic planning, hierarchic and sequential/parallel planning). This approach is mainly used for high level tasks and is employed by Marvel[Kai90], Merlin[Emm91], Grapple [Huf88] ,Agora PM[Bis88].

Marvel [Kai90] is one of earliest PM systems based on production rules to allow the modeling and control of process executions. A rule is composed of: a precondition and a postcondition that are defined in terms of attributes of objects in the object base; and an operator part used to integrate tools. Marvel manages rule execution by backward and forward reasoning. The application of these two mechanisms together is called opportunistic processing. The main innovation of Marvel is to introduce the concept of strategy[Kai90]. A strategy is a set of definitions (rules, tools, object and relations type) which can be imported or exported. Until now only menial software development activities have been experimented, and more complex activities which might involve many users and require task decomposition are not handled. In [Kai90] the authors have declared that the application of the opportunistic processing mechanism may render the object base inconsistent!

Merlin[Emm91] provides two types of production rules to represent software process with different execution mechanisms. One rule type is used to model the declarative part of process definition. Rules of this type are executed by backward reasoning. Other rule types are used to model procedural knowledge; they describe tool activation ordering, the post condition, and are executed by forward reasoning. It reduces the inconsistency problems of Marvel opportunistic processing. **Grapple**[Huf88] extended Marvel formalism to support task decomposition, i.e. a rule may be composed of other rules, in this way a high-level task may be decomposed into a set of subtasks. **Agora-PM**[Bis88] uses goal and constraint formalism (instead of production rules) to model task building, loading, and running for heterogeneous parallel systems.

3.3.2 Process programming

Process programming is an approach to software process modeling proposed by Osterweil [Ost87]. Here, the complete software process is defined as a meta-program. It is described by means of a formal language, which is written by the environment administrator before process activation. This description is considered as a specification of how a development process is to be conducted. Arcadia[Sut90] and Triad[Sar91] are examples of this approach. Both systems have extended the Ada language with new capabilities for supporting software processes. The main drawback of this approach is that no algorithm of a particular software process can be described completely in advance.

The

Arcadia project aims to build a process programming environment, based on a prototype Ada-like process programming description language, called Appl/A [Sut90] and supported by an object base [Pen91]. The principal extensions of Appl/A over Ada are relations among software artifacts, trigger upon relation operations, integrity semantic constraints on relation, and some transaction constructs. Software derivation tasks are embedded in relation definitions, and are automatically executed after software change. Triggers are used to propagate updates on relations. The triggers, constraints, and transaction statements are still in specification state.

Triad is another research prototype system heavily influenced by Arcadia ideas. Software development policies are described by an imperative language called CML (Conceptual Modeling Language) which is also based on Ada. CML is composed of: 1) an object oriented semantic data model which allows trigger definition on data types; 2) a tool model; 3) user model describing the different role types played by the user team; and 4) an activity model. The activity model is used to describe the activity hierarchy, and how each activity may be performed. CML provides primitives to synchronize parallel activities.

Arcadia has a more advanced data model than Triad, and semantic constraints may be more easily described and handled. However, the activity model of Triad is more suitable for describing software development policies than Arcadia, because activity synchronization and decomposition are explicitly modeled and executed. Unfortunately Triad has been discontinued, and no practical results have been produced.

3.3.3 Active data bases

The underlying database embeds Event-Condition-Action (ECA) rules which models the development activity. When an event occurs the action is executed if the condition is satisfied. Adele belongs to this group. The next section gives an example of ECA rules in Adele.

4 Adele 2: an example of PSEE

The Adele environment is composed of a software engineering database Adele-DB, a configuration manager and an activity manager.

These three components are described and integrated in the environment using the Adele language. This language provides ways of defining the static and dynamic aspects of the environment. The static aspects are modeled by the Data Model (ERA extended with OO), while the dynamic aspects (behavior model) are modeled by an event-action mechanism associated with a simple imperative language.

4.1 Adele data model

In the Adele data model the basic entity belongs to the "object" type. Each entity has a type descending from the "object" type. Attributes, relationships and triggers may be associated with each object.

This data model supports complex objects called **aggregates**. An aggregate is an object related with its components by relationships; the aggregate semantics is defined by the relationship behavior, which is user defined (see below). That way almost any kind of aggregate with any behavior and consistency constraints can be defined. However for efficiency reasons a special case of aggregate, called **hard aggregate**, since used for the versioning of objects, is hardwired in the system. Hard aggregate components are not sharable, their existence depends on the existence of the aggregate; all the characteristics of the aggregate instance (attributes, relationships, constraints, methods, rights) are inherited by each one of its components.

Special attention is paid to relationship management and control; they have a significant role to play in a SEE, where objects are strongly inter-related. For example, for the relation "X program depends on Y interface". As for object instances, relationships may have attributes and constraints.

4.2 Adele behavior model

The dynamic aspects of the environments constructed on the Adele kernel (e.g. environment policies), are defined in the Adele's language using trigger rule formalism. Adele triggers take the following form:

```
ON event DO Action ;
```

Where "event" is a predicate over the system state, object state and the activities underway(query, navigation as well as changes) occurring on objects.

```
EVENT delete = [ command = rm ] ;
```

An Action is a program in the Adele Language. An Adele language instruction can be a logical expression, an Adele command or a Unix command. This language is a simple imperative language, tailored to access Database information and navigate easily through arbitrary relations. It is a meta-substitution language (late binding of parameters and variables) that looks like the Unix shell, except that variables are multi-valued attributes, with provision for complex query and operator set.

Trigger is a basic mechanism, useful mainly for maintaining consistency constraints. This mechanism becomes very powerful when integrated in both the data model and the recovery mechanism.

Triggers and the data model

Triggers are defined into the type definition of objects and relations, and thus structured and inherited along the type hierarchy. A trigger defined on an object type is executed if the event is true for an instance of the type or for an instance of any subtype. Triggers cannot be overloaded, they are mandatorily inherited by subtypes.

Triggers and transactions

In Adele, users can define methods (i.e. action associated with a given object type), as well as commands (i.e. actions not related with any particular type).

Triggers are used to check the consistency of such methods/commands. Some triggers will be executed before the action, acting as pre_conditions, others after the action, as post_conditions. Since triggers are (originally) intended to enforce consistency, any inconsistency found by a trigger must be able to undo (roll-back) the action. Thus for any action the following instructions will be executed:

```
PRE list of triggers
        Action (Command or Method)
POST list of triggers
```

The whole operation is always a single transaction, even if the triggers or the action call other actions. The execution of primitive "ABORT", anywhere in a block (PRE/Action/POST) will undo everything that was done in this block. After the transaction validation; "AFTER" triggers are executed; they are used to execute an action when sure that the transaction succeeded, as for instance sending notifications, or to execute new actions whose failure must not undo the main action. If the transaction failed, "ERROR" triggers are executed.

Thus for each object *and relation type*, there are five blocs:

```
PRE    list of triggers
METHOD list of methods
POST   list of triggers
AFTER  list of triggers
ERROR  list of triggers
```

The ordering of triggers of the same kind (all PRE triggers or all POST triggers) is performed on the basis of the priority declared in the event definition.

4.2.1 An example

Figure 1 presents an example of Adele's language capacities for defining a simple environment policy:

In this *body* type description we find in lines 1 the definition of attribute *lines* which represents the number of lines in the body. *Lines* is declared *COMP* which means the value provided at instantiation is not the attribute value but the program that, when executed, will return the real attribute value. In line 9 the value of line is the result of the execution by Unix shell of *wc -l !filename* i.e the number of line in file *!filename.*

Line 2 is a pre-condition which specifies that if the event `delete_official` occurs, the command which triggered this event must be aborted. Event *delete_official* in defined line 12 occurs when the command `delete` is applied to an official body (i.e. an object body with attribute state equal to official). Line 3 expresses a post-condition on event `replace_body_c` defined in line 13. When the command `replace` is applied to a c program body (an object with the attribute language equal to c) this program must be compiled. If compilation is successful (line 9) the binary object is recorded with its source code (line 10) and the line numbers of the source object is computed and recorded (line 11).

The relation `comp` relates a configuration with its components. Before replacing a component of a configuration (line 5), the number of lines of the configuration (`!O` refers to the origin of the relation i.e. the configuration), is reduced by the number of line of the component (`!D` refers to the relation destination i.e. the replaced component, `!DD%lines` is the value of attribute lines of the component); after the replace command (line 7), the actual number of line of the component is added to the number of lines of the configuration (line 8). That way, the number of line of all configurations is always up to date and recursively.

4.2.2 Structuring and customization: partitions and sub-projects

The Adele-DB is rather general, it is suitable for various projects and an adaptation is produced for each of them. For large projects, sub-projects can be highly

```
TYPEOBJECT body ;
   DEFATTRIBUTE
      lines COMP = INTEGER;
   PRE ON delete_official DO ABORT;
   POST ON replace_body_c DO
        "store_binary %name" ;
END body;

TYPERELATION comp ;
   PRE  ON DEST replace_body_c DO
        "modify_attr !O -a line-conf = %line-conf - ~!D%lines" ;
   POST ON DEST replace_body_c DO
        "modify_attr !O -a line-conf = %line-conf + ~!D%lines" ;
END comp ;

DEFACTION store_binary;
   IF "cc -c !filename" THEN
     {"replace %name -do" ;
      "modify_attr %name -a lines = \"wc -l !filename\"" } ;
END store_binary;

DEFEVENT
   delete_official = [ !command=delete, state=official ];
   replace_body_c =  [ !command=replace, language = c ];
END
```

Figure 1: An example of the Adele language

independent and therefore may have different characteristics. For example, consistency constraints, languages or tools can differ for both the kernel and some sub-systems; for configurations in maintenance and those in development. A single data schema is not sufficient, even if most of the definitions are shared by the different sub-systems. In order to be able to define multiple data schema, the partition notion is provided.

A **partition** is a sub database (a set of software component instances) that share the same data schema. Partitions are organized in a partition tree, where the project is the root. Each partition describes the sub-schemas to be used by its components. This is different from PCTE where an SDS (Shared Definition Schema) is a view defined by a sub-set of types, but the view is always applied to the database instances. Each partition inherits definitions from its ancestor partitions, and each partition may modify (refine or enlarge) inherited definition. It is important to note that the partition concept is different from OO inheritance: a partition is not a refinement of its ancestor partition (or parent schema) but a customized schema using an overloading mechanism.

For partition P0 father of partition P1, every type defined in P0 and not defined in P1 is inherited in P1 without change. A new type can be declared in P1; if the same type T is defined in both P0 and P1, the OVERLOAD key-work means that only the new definition of T is valid, otherwise the same inheritance mechanism is applied between both definitions as between a type and its sub-types.

This mechanism allows to define P1 (a new partition) only describing the differences relative to P0 (its embedding partition).

It is not a classic inheritance mechanism since P1 can redefine (almost) P0 definitions freely, not just refine them; however some invariants must hold between the definitions of the same type: the list of its super-types and the domain of its attributes (i.e. integer, string, enum, boolean..) must remain constant.

Adele allows the association between the partition concept, which defines the relationship between the same type in different partitions, regardless of it super-types, and subtyping concept which defines relationship between a type and its sub-types in the same partitions, regardless of super-partitions. Each partition contains a set of type definitions, the ones defined locally and the ones inherited, taking into account the overloading mechanism and then the subtyping relation.

A partition plays a double role: it allows the definition of what is common between a set of objects (its common schema) and provides an abstraction mechanism since a partition is an aggregate.

A **sub-project** is a partition with two additional properties: (1)it defines an independent name space for objects, and (2) it is the unit for network distribution; it is a clustering feature.

4.2.3 Data model evolution

It is easy to add new object or relation types to a database; but very difficult to modify existing types. We choose the technical constraint to enable "any" modification of the schema. Objects may become inconsistent, but they are not deleted and must remain reachable. In Adele, each object is associated with its control block in order to keep enough information to handle it; even if the object is inconsistent regarding its type definition, it remains manageable. In this case, Adele just sends warning messages. However, work is underway on schema evolution in order to allow the definition of multiple schemata, with dynamic substitution of schema.

5 Conclusion

We have addressed the different trends in SEE, emphasizing the evolution toward more powerfull data modeling and the integration of process models. We believe that this last aspect (process models) will represent a major step forward in the field by the production of Process centered SEE (PSEE).

Simplifying a bit, we have shown that current state of the technology is to provide either

- monolithic CASE with embedded tools, services and policies;
- general platform supporting only data modeling.

We believe the next step will be to provide platforms integrating data modeling with process modeling, with emphasis on tool integration. These platforms are currently addressing some of the following aspects:

1. Machine and OS independence
2. Tool and Application writer support
3. Tool inter-operability
4. Case builder support

Monolithic CASEs address only Application writer support, Systems like PCTE are addressing points 1 and 2, Softbench points 2 and 3, Adele points 3 and 4. PCTE proposes a system supposed to replace the operating system (!), and thus provide a set of service for all OS features. However the real progress is the Object Manager System (OMS), allowing tools to share data schemas. Tool inter-operability needs more than data sharing, the Softbench Broadcast Message Server is an example of tool integration by control; Motif as a standard for User Interface integration.

Adele does not try to provide an interface for application writers but for CASE builders. Its purpose is to allow easy definition and building of SEE including services for team support. Tool integration, inter-operability, Work Environment control, team support and policy programming are its primary goals (i.e point 4). Adele addresses also point 1 and 2 by providing a PCTE interface (the Adele OMS is powerfully enough to support PCTE OMS), and point 3 by its trigger language and the integration of BMS.

However it is clear that considerable progress must be made in several directions:

- Cooperating heterogeneous and distributed repositories,
- Cooperating Cases (not only simple tools),
- Process and policy programming.

For the time being, we believe that the Adele integration of features until now usually independent like ERA data bases, OO concepts, Triggers, Tool control, Activity control is a real progress in the field of SEE, and a step toward PSEE.

References

[Bel91a] N. Belkhatir; J. Estublier; W. Melo. Adele 2: A support to Large Software Development Process. In [Dow91].

[Bel91b] N. Belkhatir; J. Estublier; W. Melo. Activity coordination in Adele: a software production kernel. In In [Tho91].

[Ber87] Ph. A. Bernstein. Database System Support for Software Engineering: an Extended Abstract. In *9th International Conference on Software Engineering*, March 1987.

[Bis88] R.Bisiani; F. Lecouat; V. Ambriola. A tool to coordinate tools. *IEEE Software*, November 1988, pp. 17–25.

[Bou88] G. Boudier; R. Minot; I. M. Thomas. An overview of PCTE and PCTE+. In *ACM SIGPLAN Notices*, 24(2):248–257, February 1989.

[Bux80] J.N. Buxton (Ed.). Stoneman: Requirements for ADA Prog.Support Env. Technical Report, U.S. Dept of Defense. Feb 1980.

[Cha90] Jean. Luc. Chauvet. Andromede et le controle du trafic aerien. *Genie Logiciel & Systemes experts.* No21. Decembre 90.

[Con90b] R. Conradi; E. Osjord; P.H. Westby; C. Liu. Software process modeling in Epos: design and initial implementation. In *3rd International Workshop on Software Engineering and its Applications*, December 3–7 1990, Toulouse, France. pp. 365–381.

[Dei90] W. Deiters and V. Gruhn. Managing software processes in the environment Melmac. In *SIGSOFT Software Engineering Notes*, 15(6):193–205.

[Dit89] K.R. Dittrich. The Damokles database system for design applications: its past, its present, and its future. In *Software Engineering Environments: Research and Practice*, K. H. Bennett (ed.). pages 151–171. Ellis Horwood Books. 1989.

[Dow91] M. Downson. *Proc. of the 1st Conference on Software Process*. Redondo Beach, CA, October 21–22 1991. IEEE Computer Society Press.

[Emm91] W. Emmerich; G. Junkermann; B. Peuschel; W. Shafer; S. Wolf. Merlin: knowledge-based process modeling. In *First European Workshop on Software Process Modeling.* Milan, Italy. May 30–31 1991. pp.181–186.

[Est92] J. Estublier and N. Belkhatir and W. L. Melo. Cooperative Work in Large–Scale Software Systems. In *Journal of Software Maintenance: Research and Practice*, K. Bennett and M. Colter (Eds.), 1992. To appear.

[Eur86] Requirements For a Tool Support Interface. PCTE+ Definition Phase Project/EURAC. 3rd edition, july 1986.

[Gal86] F. Gallo; R. Minot; I. Thomas. The Object Management System of PCTE as a Software Engineering Database Management System. In *2nd ACM SIGSOFT/SIGPLAN.* pages 12-15, 1986.

[Huf81] K.E. Huff. A Database Model for Effective Configuration Management in the Programming Environment In *5th Int. Conf. on Soft. Engineering*, March 1981.

[Huf88] K. E. Huff, V. R. Lesser. A plan-based intelligent assistant that supports the software development process. In *ACM SIGPLAN Notices*, 24(2):97–106, February 1989.

[Hud87] S. E. Hudson, R. King. Object-Oriented Database Support for Software Environments. Proc. ACM Sigmod 1987

[Lec89] C. Lecluse, P. Richard The O2 Database Programming Language *Proc. of the 15th VLDB Conference*, Amsterdam, The netherlands, August 1989.

[Kai90] G. E. Kaiser; N. S. Barghouti; M. H. Sokolsky. Preliminary experience with process modeling in the Marvel software development environment kernel. In *23th Annual Hawaii International Conference on System Sciences.* January 1990, Kona, HI. pp. 131-140.

[Kim88] Won Kim, Nat Ballou, Hong-Tai Chou, Jorge F. Garza, Darell Woelk Integrating An Object-Oriented Programming System with a Data Base System OOPSLA'88 Proceedings September 25-30, 1988.

[Leb84] D. B. Leblang, R. P. Chase. Computer-aided Software Engineering in a Distributed Workstation Environment. *ACM SIGPLAN Software Engineering Symposium On Practical Software Development Environments.* pages 104-112, 1984.

[Kim91] W. Kim; N. Ballou; J.F. Garza; D. Woelk. A distributed object-oriented database system supporting shared and private databases. *TOIS*,9(1):31–51, January 1991.

[Mar86] K. Marzullo, D. Wiebe. Jasmine: A Software System Modeling Facility. In *SIGPLAN Notices* Vol.22, No1, Jan. 1987.

[Ost81] L. Osterweil. Software Environment: Research Direction for the Next Five Years. *IEEE Computer*, Vol. 14, No 4, April 1981.

[Ost87] L. J. Osterweil. Software processes are software too. In *9th International Conference on Software Engineering.* March 1987. Monterey, CA.

[Pen91] M.H. Penedo. Acquiring experiences with the modeling and implementation of the project life-cycle process. In *IEE Software Enginneering Journal*, vo6, no5, September 1991.

[Pri86] R. Prietro-Diaz, J. M. Neighbors. Module Interconnection Languages. *Journal of Systems and Software*, 6,pp 307-334, 1986.

[Obe88] P. A. Oberndorf. The commom Ada programming support environment (APSE) interface set (CAIS). *IEEE TOSE*, 14(6):742–748, June 1988.

[Rey88] M. J. Reynier. Utilisation de l'environnement palas dans le cadre de la realisation industrielle d'un logiciel critique temps reel embarque. In *Le Genie Logiciel et ses applications.* Toulouse, France, 5-9 Decembre 1988.

[Sar91] V. Venugopal, S. Sarkar. A language-based approach to building CSCW systems. In *24th Annual Hawaii International Conference on System Sciences.* Kona, HI, 1991.

[Sel88] T. Sellis et al. Implementing Large Production systems in a DBMS Environment: Concepts and Algorithms. In *ACM SIGMOD 1988.*

[Sch88a] R.W. Schwanke, G.E. Kaiser. Living With Inconsistency in Large Systems. *Int. Workshop on Soft Version and Conf Control.* January 1988, Grassau-FRG

[Sno86] R. Snodgrass, K. Shannon. Supporting Flexible and efficient Tool Integration. *IFIP WG2.4 International Workshop on Advanced Prog. Env.*, Trondheim Norway, June1986. Springer Verlag(ed.) LNCS 244, Feb. 1987.

[Sun88] *Network Software Environment: Reference Manual.* Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043, USA, part no: 800-2095, March 1988.

[Sut90] S. M. Sutton, D. Heimbigner and L. J. Osterweil. Language constructs for managing change in process-centered environments. In *ACM Software Engineering Notes*, 15(6):206–217, December 1990.

[Swi86] D.C. Swinehart, P.T. Zellweger, R.J.. Beach, R.B. Hagmann. A structural View of The Cedar Programming Environment. *ACM TOPLAS*, 8(4):419-490, October 1986.

[Tho91] I. Thomas. *Proc. of the 7th International Software Process Workshop.* San Francisco, CA, October 16–18 1991. IEEE Computer Society Press.

[Tic82] W.F. Tichy. A Data Model For Programming Support Environment and its Application. *Automated Tools For Information Syst. Design and Dev.* A. I. Wasserman (Ed.), North Holland Ed., Amsterdam-1982.

[Unl89] R. Unland, G. Schlageter. An Object-Oriented Programming Environment for Advanced Database Applications. *COOP Advanced Databases Applications.* May/june 1989.

[Zdo86] B. Zdonik. Version Management in an Object-Oriented Database. *IFIP WG2.4 Int. Workshop on Advanced Prog. Env.*, Trondheim Norway, 16-18 June1986. Springer Verlag(ed.) LNCS 244, Feb. 1987.

[Zuc91] J.-D. Zucker. ALF: accueil de logiciel futur. In *Proc. of the 5th Conference on Software Engineering Environments*, Aberystwyth, UK, March 25–27, 1991. pp. 21–52