# Program Families: Some Requirements Issues for the Process Languages

Gianpaolo Cugola, Carlo Ghezzi
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza L. da Vinci 32
20133 Milano, Italy
{cugola, ghezzi}@elet.polimi.it

## Abstract

*In this position paper we try to address the workshop theme by discussing some issues concerning process support for product families. In particular, we discuss how the particular problem of supporting product family developments affects the notation used for process representation and enactment. Based on our current understanding of the problem, our conclusion is that the development of program families does not introduce new requirements, but stress some of the requirements that are intrinsic to most software processes to their extreme. That is, they do not demand for new language mechanisms, but require specific processes to be put in place.*

## 1. Introduction: product families

A product family is a set of products that have common properties. Products are clustered in a family because it can be convenient to analyze, design, and manage the family as a related set of elements, thus stressing the common properties of the set, rather than concentrating on each member of the family separately. Examples of a family are the different versions of compilers, operating systems, productivity systems. Members of the family may differ because, for example

- the set of functionalities provided by each family member differs;
- the different members of the family runs on different target machines.

Most of the methodologies and technology available today to support software development focus on developing a single software product. Increasingly, however, software developers deliver product families rather than individual products and need methods to master their development as well as support to the production process.

In this position paper we will try to understand:

1. how should a rational design process be structured for a product family;
2. how in practice most developments are structured and why.

Our goal is to use the results of the analysis of points 1 and 2 to derive requirements for a process language. In particular, point 1 will help us understand the principles upon which an ideal language for process support environment should be founded. Point 2 will help us understand additional important requirements that would improve usability of the resulting environments.

## 2. Development methods for product families

The concept of program families was first introduced by Parnas in a seminal paper published in 1976 [1] in the context of design methodologies. Although this paper is now 20 years old, the lessons drawn from it are still valid. Parnas argues that

1. programs *inevitably* exist in many versions (*products*);
2. a rational design process should address the design of all versions, (i.e., the *product family*).

Parnas does not distinguish between different versions of the same product developed to increase product functionality and/or eliminate bugs and different versions that represent different products of the family. The underlying design methodology problems boil down to the same issues in both cases.

Parnas contrasts the classical method of developing products with what would be the ideal method to develop families. The classical method, called *sequential*



0-8186-7725-2/97 $10.00 © 1997 IEEE

**Figure 1a. The sequential completion method**

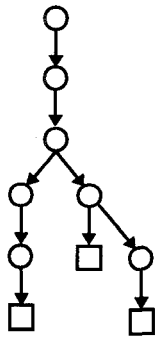

**Figure 1b. The ideal method**

*completion (SC)*, works on a particular member of the family, which is completely developed. The next member of the family is then developed by modification of the previous. The method can be abstractly represented as in Figure 1a, in which the design process is represented as a linear sequence of design decisions (represented by a circular node) until a product (represented by a square node) is reached. Starting from the product, one can then start a new chain that eventually yields another product.

The alternative method, called here the *ideal method (IM)*, requires a special effort from the designer to try to identify all possible family members since the very beginning of the development. In the case of a single product whose evolution must be supported, this means that all future possible changes of the product are to be anticipated. In the case of a multi-product family, an effort must be paid in understanding exactly the requirements of all member products. Design should proceed on the family as a whole, trying to delay any decisions that discriminate among different family members to the possibly latest point. The ideal method can be abstractly depicted as in Figure 1b. Branches in the tree representing design decisions where the family of products is partitioned into subfamilies. Each node of the tree is associated with a state of the process. Obviously, the tree does not imply that all members are developed in parallel: one can traverse the tree depth-first to develop some member products. It is important, however, that whenever a new member is to be developed, design for the new member starts from the node of the tree that discriminates the subfamily in which the new member belongs from the existing products. One should not obtain the new member through modification of another product, but design should start from the appropriate design node (i.e., from the appropriate process state).

## 3. Requirements for the process language

The discussion in Section 2 shows that specific processes must be put in place to support product families. We do not try to identify such specific processes here. Rather, we will try to find the main requirements for the process language that can be derived from the sketchy discussion in Section 2. They are:

1. *Persistence of process states.* Process states must be recorded in order to make it possible to move back on the process state hierarchy to develop new family members;
2. *Rationale associated with process states.* It is important that process states do not include just data and artifacts, but also the rationale of design choices. The process language should allow for reasoning mechanisms to be associated with this information. These inferential mechanisms should help developers in deciding the past process state that is best suited to start the development of a new member of the family;
3. *Object-Oriented type system.* The hierarchical nature of the method shown in Figure 1b demands for an object-oriented type system for both process data and process fragments. Both inheritance (in particular, the opportunity of defining abstract data types and the ability of adding new features to a parent type) and genericity (i.e., providing type-parametric abstractions), as well as polymorphism and dynamic binding, are necessary to model the hierarchy of Figure 1b in a natural way;
4. *Degrees of concurrency.* Individual product development may be structured as a set of concurrent activities (concurrency in the small). Furthermore, in a product family development, it should be possible to structure the various product developments as concurrent macro-activities (concurrency in the large).

We will now try to identify additional requirements that may arise by examining how in practice product families are developed and why. As we observed, although Parnas' paper is 20 years old, practical developments diverge from the ideal method more or less severely. An overzealous approach to the development of product families would consider any deviation from the ideal method as illegal. Process technology would then be viewed as a way to precisely define a model of the ideal method and a mechanism to enforce its use. There are several reasons, however, suggesting that diverging from the ideal method necessarily happens in reality. The ideal method can only be approximated and used as a guideline, but it cannot be followed as a dogma. This fact was recognized in another seminal paper [2], which acknowledges that any ideal model (e.g., a top-down method) can only be used as a general guideline during the process or it can be used *a posteriori* as a way of providing a rational description of the result of the process. Most of the comments in this paper hold also in the case of product families, for the following (non exhaustive) list of reasons:

- an organization may decide to generalize a product into a family of products after commercial success of the product, i.e., the family concept is not present since the conception of the product;
- it may be impossible to anticipate all different family products, especially in the case of innovative products;
- the need for early delivery of a family member, which may be dictated by serious market concerns, can bias early design steps.

In other words, the ideal method represents only an utopia. The processes used in reality to develop product families, and the models of these processes, have to tend to this best solution. The main consequence of the fact that real processes always diverge from the ideal method is that during the development of a product family it is often necessary to deviate from the modeled process to cope with unexpected situations.

The problem of deviations of the actual process from the modeled one is a problem common to all the software processes [3], but becomes a really crucial issue in the case of product families.

Dealing with deviations may require a whole spectrum of support features by the environment and by the process language:

- ability to tolerate temporary deviations without the need of modifying the process model;
- ability to deal with inconsistencies that may arise because of deviations;
- ability to change the process fragment instances as they are executing;
- ability to change process definitions.

In conclusion, of all the requirements we have been able to identify for process languages, none is completely new because of product families, and most are provided by existing process languages. We believe, however, that the requirements we listed are particularly important in the case of product families. For example, the usefulness of an OO type system has cogently been discussed in [4]. An OO type system simply becomes a vital requirement in our case. A precise understanding of the features that are particularly stressed in the development of product families is fundamental to guide the implementation of a specific process support system.

## References


[1] D.L. Parnas, On the Design and Development of Program Families, *IEEE Trans. on SE*, SE-2, 1, pp. 1-8, 1976.

[2] D.L. Parnas, P.C. Clements, A Rational Design Process: How and Why to Fake It, *IEEE Trans. on SE*, SE-12, 2, pp. 251-257, 1986.

[3] G. Cugola, E. Di Nitto, C. Ghezzi and M. Mantione, How To Deal With Deviations During Process Model Enactment, *Proceedings of the 17th International Conference on Software Process (ICSE17),* Seattle, Washington, USA, pp. 265-273, April 1995.

[4] W. Emmerich, W. Schäfer and J. Welsh, Databases for Software Engineering Environments - The Goal has not yet been attained, *Proceedings of the 4th European Software Engineering Conference (ESEC93),* Garmish-Partenkirchen, Germany, pp. 145-162, September 1993.