Developing Software Families

Silva Robak

Institute of Organization and Management
University of Zielona Góra
ul. Podgorna 50,
PL-65-246 Zielona Gora, Poland
S.Robak@iiz.uz.zgora.pl

Abstract: There is a lack of a systematic, generic, domain independent object-oriented software engineering process for software familes. In such process a system family should be treated as a whole (opposite to multiple products maintained separately) and the reuse should be planned with assets developed for reuse. Particularly important is the embedding the domain engineering methods within object-oriented methods, especially the feature modeling for modeling variability. In the paper the problems associated with developing software familes and handling variability are presented and some aspects of generic development process for software families are introduced.

1 Introduction

The object-oriented paradigm has brought the new concepts like classes and objects, inheritance, dynamic polymorphism and dynamic binding to software engineering. Despite the advantages of above concepts the object-oriented software paradigm has not reached productivity, which had been expected on the area of reuse, adaptability and management of complexity [We95]. The object-oriented approach is concerned with a development of a one system at a time and mainly supports reuse of assets (especially a code), in the next versions of a single software product. A software *asset* is a description of some partial solution (e.g. component, design document, model or knowledge) that engineers use to create or modify software products [Wi96]. The object-oriented reuse concepts embrace the ideas like class libraries (e.g. STL, [Br98]), frameworks [Jo97], patterns [Ga95] and components [Sz98] (e.g. COM+, CORBA, EJB). In the object-oriented approach a single system is developed "with reuse", and that is an opportunistic, small-grained reuse and the object-oriented software projects tend to excess the foreseen budgets and delivery times [Bo00].

Besides the object-oriented also another approaches towards improving efficiency of the software development are known. There are generative techniques and the new software paradigm the Generative Programming (GP) [CE00]. Further, based on OOP raised the Aspect Oriented Programming (AOP) [Ki97] and the Subject Oriented Programming

(SOP) [Os95]. Furthermore, based on the ideas of the Domain Engineering and Application Engineering (see below), there are methods like Reuse-driven Software Engineering Business (RSEB) [JGJ97] Organization Domain Modeling ODM [Si96], etc. Feature driven techniques like Feature Oriented Domain Analysis (FODA) [Ka90], FORM [Ka98], and FeatuRSEB [GFD98] are other ways using the advantages of the feature modeling. Features are essential characteristic of systems within a system family and are organized in different kinds of diagrams containing hierarchies of feature trees.

A development of a group of systems built from common generic software assets is a goal by building software product-lines upon the product families. The fundamental reason for creating program families has been already presented in early works by Dijkstra [Di72] and by Parnas [Pa76]. *Program families* were defined by Parnas as "sets of programs, whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members" [Parnas76]. Building the sets of related systems helps to achieve the remarkable gains in productivity and improves time-to-market and product quality [CN99]. A product family may extend across several domains. According to Software Engineering Institute (SEI) a *software product line (PL)* is "a set of software-intensive systems sharing a common managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of a core assets in a prescribed way" [CN99]. In the paper the terms software product line and software family will be used interchangeable, but it has to be emphasized, that a product line bears always some *business goals.*

There are some known process models like: the waterfall approach, evolutionary development, formal transformation and system assembly from reusable components [So95]. Current accepted object-oriented software development process models are no more waterfall-like, but spiral, iterative-incremental process models, e.g. the quasi-standard The Unified Process (UP). The explicit consideration of the risk is the most important distinction between the spiral models and the other process models. Software development process for product lines is alike a software development process for a single system. However, there are some significant differences (see below), caused by the strategic, large-grained reuse.

There are three basic activities by developing a software PL:
- Domain Engineering (DE),
- Application Engineering (AE),
- Management.

The three fundamental, interrelated activities: **Domain Engineering** i.e. platform (i.e. generic architecture) development, Application Engineering i.e. product development from the platform, and Management on technical and organizational level are presented at Fig. 1-1. As shown, there is a strong feedback between reusable assets and products. The Management on technical and organizational level is a part connecting the whole together. The reuse is planned, enabled and enforced - as opposed to opportunistic small-grained reuse by development of a single software system. Product family assets are developed *for* reuse, in contrast to a case of a single system developed *with* reuse. All

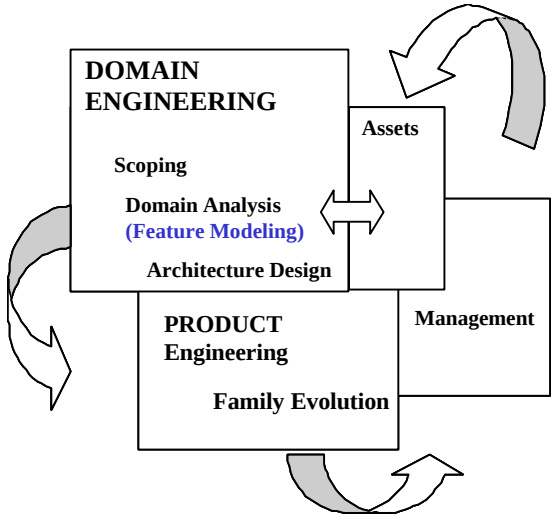

Fig. 1-1. Basic Activities By Developing System Families with Domain Engineering.

family members together are regarded as a whole. By the development of a single system there are separately maintained releases and versions of a single product. In software family (especially in product-line context) the evolution of products is planned and all (also the early) versions of the products are feasible family members. The product line architecture is merely one of the core assets in the repository and contains components, which will be assembled in a prescribed way according to the architecture and the given production plan. Important is that in system family the architecture and its components should include the built-in variability mechanisms allowing instantiating the multiple family members.

In the following sections variability modeling with feature models is summarized. Development of software product lines with RUP and a SEI framework will be nearer considered in the beginning of Section 3. Then a generic development process, as a framework with high degree of built-in variability is presented. The last section concludes the paper.

2 Variability Modeling With Feature Models

Different products as also product evolution in time cause common and variable products characteristics. Software family is characterized by [WL99] as "a set of items that have common aspects and predicted variabilities." According to Weiss and Lai, the term *variability* is defined, as “an assumption about how members of a family may differ from one another and *commonality* is “an assumption that is true for all members of a family”. The feature models introduced in FODA provide an abstract, independent and

concise representation of common and variable parts of the software family. Determining, what will constitute the common parts and what the variable parts in a product-line, has more strategic then technical nature and can change over time. Commonality constraints the size of the family, but is an important way for the future standardization within an organization and allows better reaching of productivity and efficiency goals. Variability (as optional or alternative features) enlarges the family size, but later at generic (i.e. parameterized) places also increases the systems' complexity.

A *feature* is a stakeholder (e.g. users, customers, developers, managers, etc.) visible characteristic of concept (e.g. system, component, etc.), which is used to describe and distinguish system family members. Some features relate to end-user visible characteristic, while others relate more to a structure of a system and system capabilities also including non-functional requirements [RFP02]. The feature model indicates the *intention* of the described concept. The set of instances described by feature model is the *extension* of the concept.

3 Developing System Families

Such typical object-oriented processes as UP, designated for a development of the one system at time bear some significant lacks, if they would be applied for the development of the software family. First, there is no distinction between DE and AE Phases. The evolution of products and the resulting feedback (for the whole family) are not comprised in the process. Useful and necessary domain analysis activities like domain scoping and feature modeling are not included. Moreover, in the RUP methodology there is a basic conflict between "use-case driven" and "architecture centric" approach. The strengths and deficiencies of the Rational UP are summarized in Fig. 3-1.

Domain engineering (DE) activities (see Fig.1-1) are impacted by the product constraints, provided styles, patterns, frameworks, production constraints and strategy, and the content of the repository of preexisting assets. Product constraints originate from the commonality and differences (variability) between the products, the product properties (current as also foreseen), as well as from the enforced use of existing standards, assigned performance limits, interfaces and quality requirements (like performance, reliability, modifiability, and security). Production constraints and strategy include the wide range of items like: accepted standards, time- to-market, the use of legacy components, underlying infrastructure, COTS, etc. Especially important is also the chosen general approach for developing the core assets i.e. assembling or generating of family members. The main DE artifacts include: the product-line scope, the core assets and the production plan for the family members. Product-line scope i.e. the products (family member) that will constitute a product-line, is defined within the scoping activity. The core assets i.e. architecture, components (also COTS) contain a process for their use in development of products including the variation requirements for variation points and methods for resolving variability. A *variation point* is a point identifying one or more locations, at which the variation will take place [JGJ97]. The inputs for application engineering (AE) form the DE-artifacts and together with the

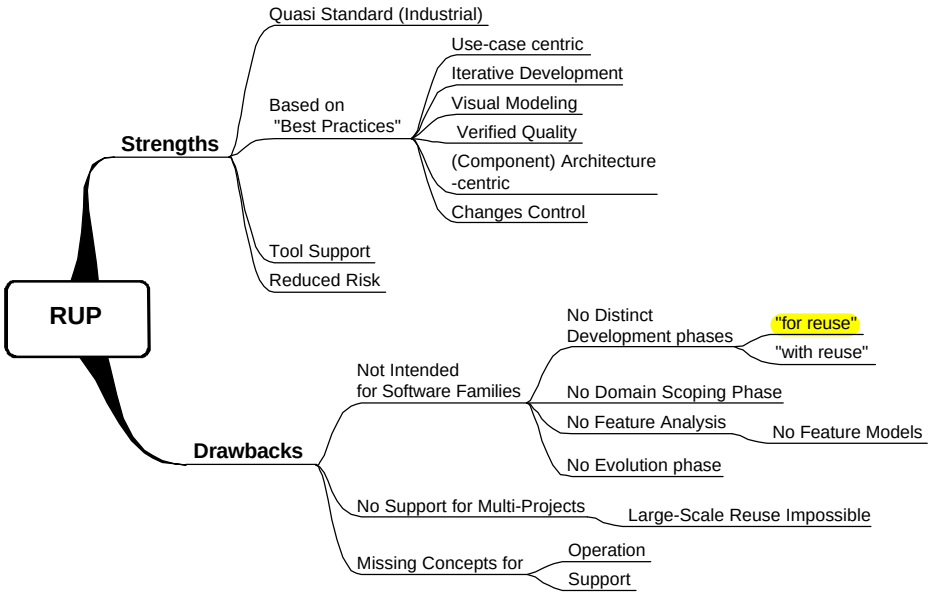

Fig. 3-1. RUP Strengths and Deficiencies.

requirements for a particular product, which are expressed as deltas or variations to an established base. AE results are the various products. Technical management is dedicated to plan the creation and evolution of core assets and products. Furthermore, the organizational management is necessary for the synchronization of the entire software family development effort.

3.1. Generic Development Process for Software Families

The SEI-Framework [CN02] (see Fig. 3-2) is a proposal of the standard framework process for software families. It also does not integrate some important (in the opinion of the author) DE items like the feature modeling and applying of the generative approaches. It has also no as connection with the Model Driven Architecture (MDA) from OMG [OMG online]. The description of the process is too intertwined to prove in the praxis. The SEI-Framework Practice Areas are depicted in Fig. 3-2. The use of the cooperating Practice Areas in form of hierarchical patterns is described in [CN02a]. Fig. 3-3 summarizes the cooperation of the proposed patterns with the main pattern – Factory. Other known processes designated for software families like PuLSE [Ba99]– Fraunenhofer IESE, TrueScope [De00], Family-oriented Abstraction, Specification and Translation (FAST) [WL99] and industrial experiences of Jan Bosch [Bo00] are narrow, oft domain dependent , specific approaches. E.g. the main drawback of RSEB [JGJ97] is the fact that the domain analysis activities are not included.

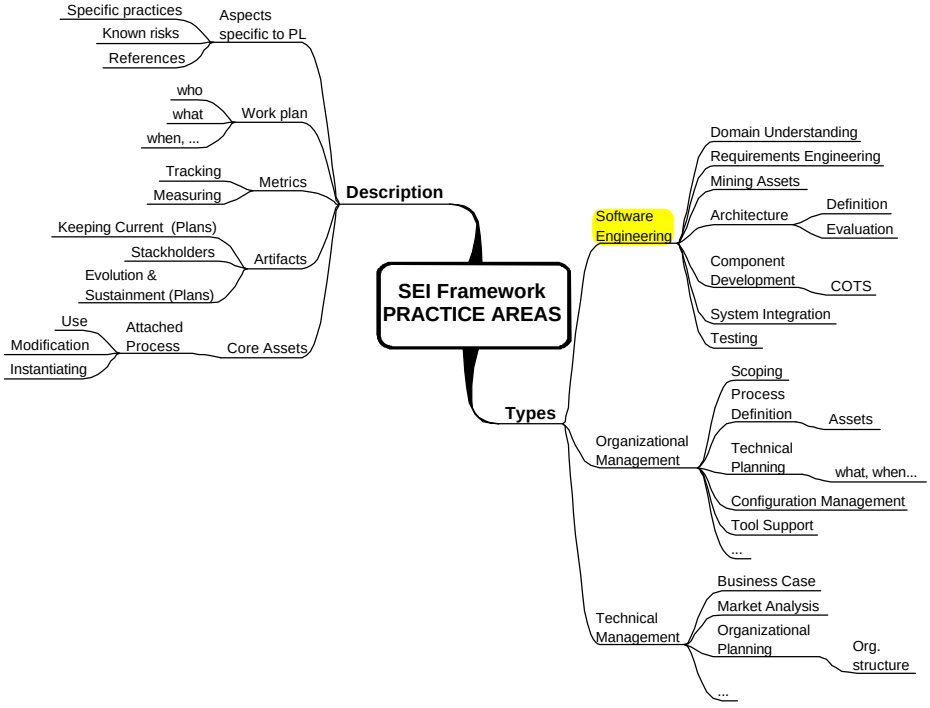

Fig. 3-2. SEI Practice Areas.

The software development process for software families has a top-down nature, and is continuous, incremental-iterative meta-process with a dual nature (as well for family as for its members), with a high degree of the built-in variability, which has to be resolved for particular family members.

The *dual nature* of the process signifies that it embraces as well the development of a whole family (generic parts) as concrete products (family members). The process components (phases) have a sequential nature: the output from one phase provides input for the next one. The top-level view of the process phases is following:

– Business Goals and Scoping,
– Domain Analysis Activities (with Feature Modeling),
– Platform Architecture Evaluation,
– Assets Integration (Components and other assets),
– Product Instantiation and Testing,
– Products Deployment and Maintenance,
– Evolution (Feedback).

Table 3-1: Process Components (phases) of Software Products Family and its Members.

Process Component	**Domain/Family**	**Product/Member**
Analysis	Business Goals & Scoping Domain Analysis (Feature Modeling)	Products Requirements Analysis, Choice of Products' Features (Feature model)
Design	Platform Architecture	Product Architecture: - Fitting of Platform Architecture, or - Development of Product-specific Architecture
Development (other Assets)	Generic Components	Product Components: - Fitting of Generic Components, - Development of Product-specific Components
Testing and Integration of Assets **Assets Implementation**	Generic Assets	Product Instantiation and Testing
Production		Product Deployment Product Operation and Maintenance
Evolution	Whole Family Evolution and Organizational Transformation	Embedding New Features

Within the phases the development is highly iterative, especially for the architecture evaluation. The partition of the process for the family and the product is depicted in the Table 3-1. Business Goals & Scoping contained in the Analysis Phase is obligatory for market oriented planning for the software product line. Changes during Products Deployment and Maintenance Phases cause feedback for the product as well as the integration of the changes into the family. Thus the process phases and their content are highly interwoven what is caused by the nature of the continuous evolution of the product family and influence of changes in one part reflected in other parts.
The *top-down* principle by the development of the product line artifacts is necessary and the only possible way of development for the product-line [Bo00]. The assets contained in the product line are created, evaluated, maintained and evolved and they have their

own life cycle that is orthogonal to the whole process for product line. The life cycle of the products is contained in the life cycle of the family.

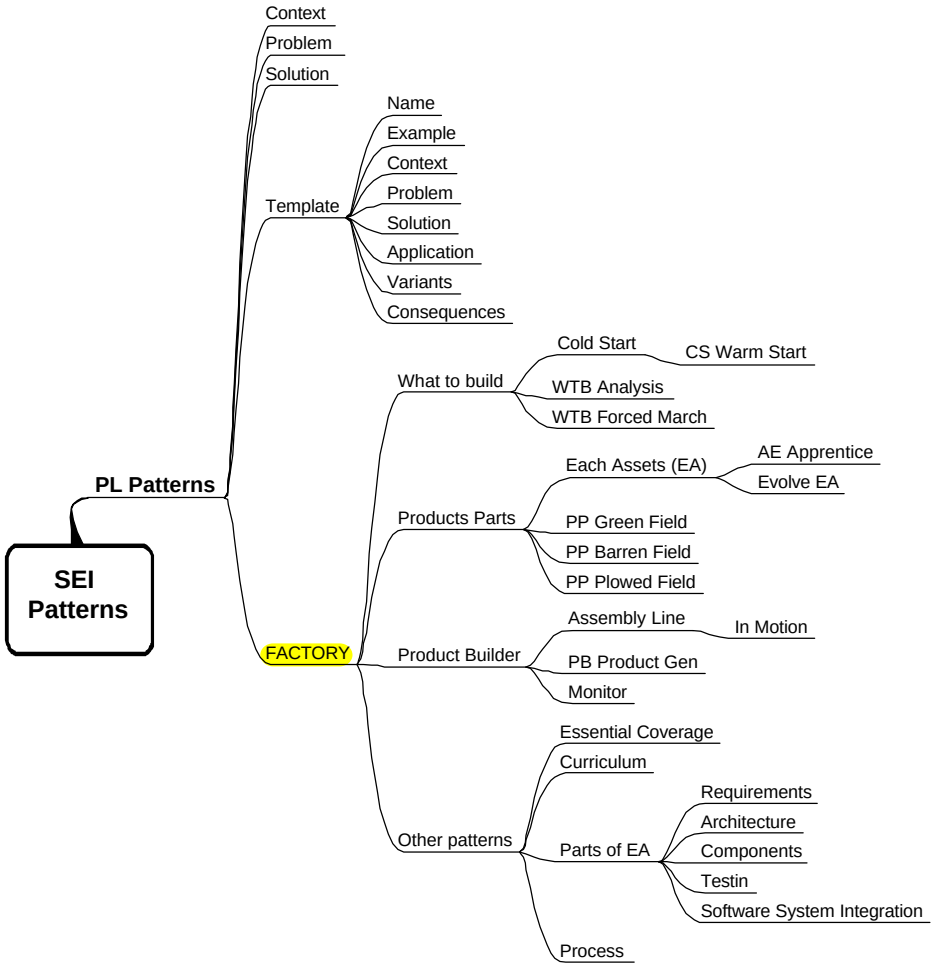

Fig. 3-3. SEI Cooperating Patterns.

The process may be seen as a meta-process with itself a high degree of variability for building its instances. The sources of *variability* in the process itself are following facts:

– Preliminary against continued product line,
– Existence of legacy systems,
– Chosen type of development,
– Available staff.

For the new software product-line (that will be possibly developed from scratch) its scope should be first determined, in contrast with the existing product-line, which scope has only to be changed. There also may be (or not) a legacy system to be integrated in

the product-line. With the chosen type of development is meant, how the core assets become instantiated, i.e. different possible ways like: mining, building, buying (e.g. COTS), or commissioning. The staff may be inexperienced in product line approach; the assets may only need modifications, not a full development.

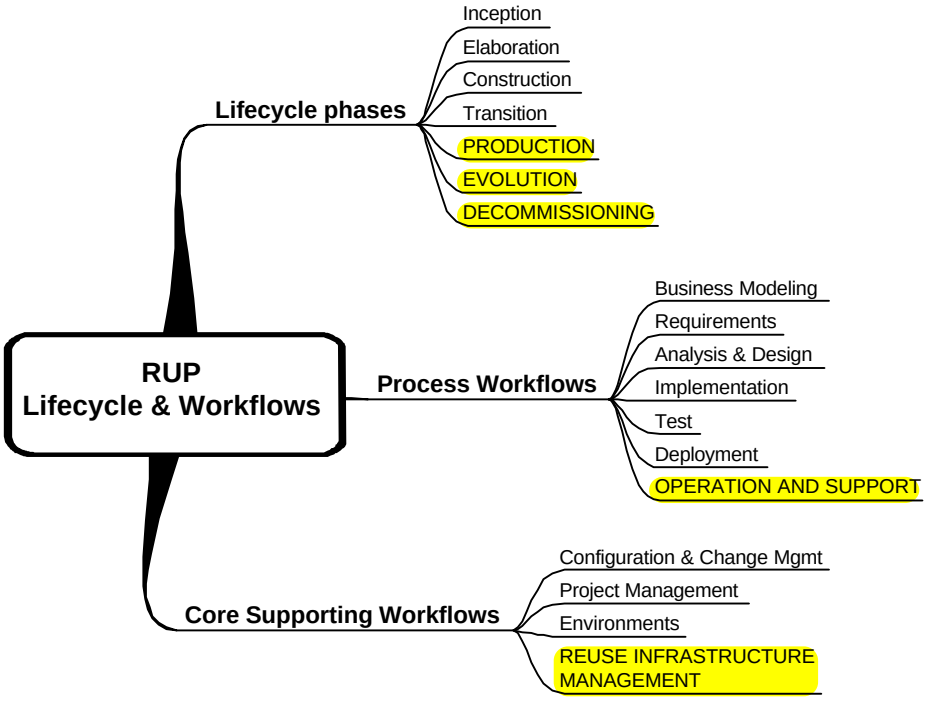

Fig. 3-4. RUP Lifecycle Phases and Process Workflows –Suggestion for Extensions.

The proposed process forms a framework into which elements may be plugged-in to enhance the capability of the skeleton, which is containing common parts for assets. The process framework includes following common parts:

- Resources assessment,
- Planning (for developing and using assets),
- Monitoring,
- Deficiency identification and resolving,
- Change control.

The sub-processes consist of specific activities required for a particular (core) asset. The common parts of sub-processes are:

- Work plan (for Tracing and Controlling),
- Tool support,

– Test Cases,
– Configuration Management.

Furthermore, family specific development includes:
– Description of variability within the asset (variation points contained in the asset and methods for resolving variability in concrete products),
– Process prescribing how to create an asset as well as how to derive products from the asset.

In the process there are also plans for evolving and maintaining the family members (products). The plans and tests are handled as other assets with contained commonality and variability.

In the Fig. 3-4 depicts the Lifecycle phases of RUP together with the Core Process and Supporting Workflows. The missing elements required for a system family are denoted with capital letters. The family (and product) life cycle is not restricted to deployment, as proposed in RUP. As long as a family exists, the products are maintained and the feedback causes the evolution - see Production and Evolution as the "last" Lifecycle phase, and Operation and Support Process Workflow in Fig. 3-5. The lifecycle is continuous and not ending with a deployment of a single product. Besides, the partition in Family and Member- Development is needed. In the Supporting Workflows the Management of the whole Reuse Infrastructure is necessary to enable the large-scale reuse opportunities.

For description of commonality variability, not only for the generative purposes, but also as a general notation for description of system (parts) containing commonality and variability the feature diagram is recommended. The description should contain mandatory and optional parts (features) and possibility for following choices: all (AND), within the alternatives as *one*-of-many (XOR) and *n*-of-many (OR) choices (e.g. as proposed in [CE00]). Embedding of feature diagrams in UML allows besides using the widely known standard object-oriented notation, the description of the composition rules within the diagrams.

4 Conclusion

The known processes designated for software families do not cover all the demanded aspects as the use of the domain analysis activities (especially the feature modeling), handling variability in the way allowing reuse above the code level, and possible wide applying of the generative approaches in the AE phase. In the paper is a contribution to improvement of the generic development process in product lines engineering (i.e. for the software system families) is presented. It integrates the best (in the opinion of the author) parts of the known in the world, existing domain-modeling methods and contributes some new improvement aspects to the process, seen itself also as a framework with common and variable parts. For the fulfillment of the generic system architecture the use of generative techniques (e.g. frame technology) is recommended, what especially supports the reuse in the evolution and maintenance of system family members.

The problems like description of the of feature model semantic (especially feature interactions) and determining generic software product line architectures according to OMG Model Driven Architecture ideas should be further investigated.

References

[Ba99] Bayer, J. et. al: A Methodology to develop software product lines. In Proceedings of the Symposium on Software Reuse. May 1999, pp. 122-131.

[Bo00] Bosch, J.: Design and Use of Software Architectures. Adopting and evolving product-line approach. Addison-Wesley , New York, 2000.

[Br98] Breymann, U.: Designing Components with The C++ STL - A New Approach To Programming. Addison-Wesley , New York, 1998.

[CE00] Czarnecki, K.; Eisenecker U.: Generative Programming Methods, Tools and Applications. Addison-Wesley , New York, 2000.

[CN02a] Clements, P.; Northrop, L.M.: Software Product Lines. Practice and Pattern. Addison Wesley, New York, 2002.

[CN02] Clements, P.; Northrop, L.M.: A Framework for Software Product Line Practice - Version 3.0 [online]. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. March 2002. Available WWW: <URL: http://www.sei.cmu.edu/plp/framework.html>.

[De00] DeBaud , J.M.: The Truescope Approach to Software Product Family Engineering. The First Software Product Line Conference (SPLC1), 2000. Denver, USA.

[Di72] Dijkstra, E.W.: Notes on Structured Programming. In: Structured Programming (O.J. Dahl, E.W. Dijkstra and C.A.R. Hoare, Eds). Academic Press, London, 1972.

[Ga95] Gamma, E. et. al: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, New York, 1995.

[GFD98] Griss, M. L.; Favaro, J. ; D'Alessandro, M.: Integrating Feature Modeling with the RSEB. Proceedings of ICSR98, Victoria, BC, IEEE, June 1998. pp.36-44.

[JGJ97] Jacobson, I.; Griss, M.L. ; Jonnson, P.: Software Reuse: Architecture, Process and Organization for Business Success. Addison –Wesley Longman, New York, 1997.

[Jo97] Johnson, R.E.: Frameworks = (Components + Patterns). Comm. of the ACM, Vol.40, No.10, 1997. pp.39-42.

[Ka90] Kang, K. et. al: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990. Pennsylvania.

[Ka98] Kang K.C. "FORM: a feature-oriented reuse method within a domain-specific architectures" in Annals of Software Engineering, V5, 1998. pp.354-355.

[Ki97] Kiczales, G. et. al: Aspect-Oriented Programming. Proceedings of ECOOP97 – 11th European Conference of Object-Oriented Programming, Jyväskylä, Finland, June 1997, (M. Aksit and S. Matsuoka, Eds.), LNCS 1241. Springer Berlin, New York, 1997.

[OM00] OMG, Object Management Group, http://www.omg.org, 2000

[Os95] Osherr, H. et. al: Subject oriented Composition Rules. In Proceedings of 10th Conference of OOPSLA'95, ACM SIGPLAN Notices, vol. 30, no 10, 1995, pp.235-250.

[Pa76] Parnas, D.L.: On the Design and Development of Program Families. IEEE Transactions on Software Engineering, March 1976. pp.1-9.

[RFP02] Robak, S.; Franczyk, B.; Politowicz K.: Extending The UML For Modelling Variability For System Families, International Journal of Applied Mathematics and Computer Science, 12(2), 2002.

[Si96] Simos, M. et. al: Organization Domain Modeling (ODM) Guidebook, Version 2.0, Informal Technical Report for STARS, STARS-VC-A025/001/00, June 14, 1996.

[So95] Sommerville, I.: Software Engineering. Addison-Wesley Publishing Company, Wokingham, 1995.

[Sz98] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley, New York, 1998.

[We95] Webster, B.: Pitfalls of Object-Oriented Development: A Guide to the Wary and the Enthusiastic. M&T Books, New York, 1995.

[Wi96] Withey, J. : Investment Analysis of Software Assets for Product Lines. Technical Report No. CMU/SEI-96-TR-010, ADA 315653. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[WL99] Weiss D.M.; Lai C.T.R.: Software Product-Line Engineering: A Family Based Software Development Process. Addison-Wesley, New York, 1999.