# An Evaluation of Domain Analysis Methods

X. Ferré
xavier@fi.upm.es

S. Vegas
svegas@fi.upm.es

Facultad de Informática –Universidad Politécnica de Madrid
Campus de Montegancedo s/n, 28660 Madrid

## ABSTRACT

Reuse of products, processes and all kinds of knowledge has been identified as a goal in Software Engineering, in order to develop reliable and high quality software systems on schedule and within budget. The duality of issues in software reuse (there are producers and consumers of reusable artefacts which might not be the same people) calls for methods as Domain Analysis to systematically build reusable elements. There are a host of Domain Analysis methods today, and nobody seems to know the reason of this variety. This paper surveys a set of relevant DA methods, with the aim of finding an explanation for the diversity of domain modelling techniques, and the type of information they represent. The results of this survey are presented, along with the conclusions obtained.

## 1. INTRODUCTION

Software reuse has been identified as a goal by several organisations [Leach97,Basili91]. Reuse of products, processes and, generally, all kinds of knowledge is an effective way to achieve the main goal in Software Engineering (SE): the development of reliable and high quality software systems on schedule and within budget [Basili91,Basili92]. One of the difficulties involved in reuse is the duality of issues: on the one hand, there are operational issues (reusing existing information to develop software systems); on the other hand, there are infrastructure issues (defining, populating and evolving the repositories of reusable information) [Rombach94,Arango91,McClure97]. This duality calls for different methods to build reusable elements.

This paper studies different Domain Analysis (DA) methods, which are used in formal reuse *to identify, organise and model knowledge about the solution of a class of problems (the class of problems is known as domain) to support its reuse among all elements in the class*. Obviously, along with the (reusable) solution, there has to be a description in each case of what the domain (class of problems) is.

Reusable elements are identified in DA during the modelling phases: data analysis (also called subsidiary modelling) and classification (also called primary modelling) [Arango94]. Methods differ with regard to the parameters used in the classification: features [Kang90,Simos91,Simos95], facets [Prieto-Díaz87,Birk97], capabilities [Cornwell96], etc, and the type of data analysed: entities and relationships [McCain85, Jaworski90,Prieto-Díaz91], context [Henninger95,Hollenbach96], etc.

There are a host of DA methods today. This variety can be seen from two opposing points of view: as an advantage (there can always be assumed to be a method that meets the particular needs of the reuser), or as a disadvantage (the question arises of whether the variety is a response to shortcomings or an overspecialisation of existing methods). Authors like [Arango94] have evaluated different DA methods, focusing on the differences in the DA process (how the domain model is obtained) rather than on other method features. For example, the comparative study performed by Arango [Arango94] on several DA methods concluded that all DA methods follow a common process, and that there are more similarities between the methods than differences. Nevertheless, these differences exist, and there must be some way of ascertaining their meaning, so that we can decide which method is the most appropriate in each situation.

This paper seeks to perform a comparative survey of DA methods similar to the study conducted by Arango, but focusing in this case on the different means proposed by the methods of developing reusable candidates.

An added problem in the construction of reusable elements is that software reuse encompasses not just software products, but also everything related to software construction; that is, process, technology, technique reuse, etc. Some approaches have recently tried to apply DA to these elements. As a result, several questions have been raised: is the concept of what DA is the same? Do they pursue the same goals? Is the modelling similar in any way?

All these questions suggest that DA methods are dependent on:

- The kind of object to be reused.
- The reuse problem to be solved: building libraries of reusable elements, systematisation of the construction of reusable elements, improvement of the reusable elements, etc.

The consequences of these dependencies are as follows. On the one hand, there are no decision-making criteria for deciding whether or not to use a method. On the other hand, although the process appears to be generic, modelling techniques do not appear to be generic enough to be able to be used for modelling reusable elements apart from software products. This is because the modelling techniques traditionally used in software systems development (functional and data decomposition, object oriented, features definition techniques) are employed in DA, which makes it difficult to apply to a more generic framework. The problem is that there are no DA specific modelling techniques that are generic enough to model software elements apart from software components or software architecture.

This paper surveys relevant DA methods, with the aim of finding an explanation for the diversity of domain modelling techniques and studying which points they use to represent reusable elements (candidate characteristics) and what these mean in each case. The concept of DA applied by each method is also analysed, as this will influence the type of information to be represented in each case.

The paper has been structured as follows: in Section 2, the DA methods classification framework is presented; Section 3 describes the parameters used in the methods studied; in Section 4, the results of the evaluation performed are presented; and, finally, Section 5 presents the conclusions drawn from the survey. The results of this survey could serve as a starting point for proposing modelling techniques that can be used for all kinds of reusable elements

## 2. CLASSIFICATION OF DA METHODS

Although there are numerous DA methods, nobody has ever tried to classify them. Nevertheless, some authors have compared different methods according to different criteria [Arango94, Wartik92].

A DA method should propose two things:

- A domain ontology, along with a taxonomy of this ontology, both of which will appear in the domain model.
- A process which, if strictly adhered to, will allow construction of the (generic) domain model.

The sequence of steps to be defined in the process will be clearly influenced by the form of the domain model. The question is, however, what should the domain model specify? According to [Arango91], a domain model should set out the information required (now and in the future) to allow reuse in the domain. Traditionally, it has been accepted that the information required for modelling consisted of identifying entities, operations, events and relationships in the domain. However, if DA is applied to other software elements, apart from software products (architectures, code, etc.) the modelling and the way models are built will have to be changed.

This means that both the domain model and the model construction process will be very much influenced by the type of element to be reused.

In so far as DA methods depend on the type of element to be reused, all the methods will have to be classified so as not to compare pears with apples. So, DA methods will be classified according to the type of element they intend to reuse. Accordingly, there are:

1. DA methods for software product reuse.
2. DA methods for software process reuse.
3. DA methods for software technology reuse.
4. DA methods for software experience reuse.

**DA methods for software product reuse** have been more widely addressed. Despite the fact that the idea of reusing code evolved into the reuse of other kinds of software products, most of the effort that has gone into reuse research has addressed software product reuse. These DA methods can be divided into four subclasses, as listed below:

1. Methods for software component reuse.
2. Methods for asset reuse.
3. Methods for software architecture/design reuse.
4. Methods for software requirements reuse.

**Software component reuse** can be seen as a search for components that supply the functionality needed by the reuser. The underlying idea is to build a software component library around the domain.

**Asset reuse** claims to be more than just component reuse, but it is not clear what, as nobody has given a precise definition of what an asset is. The basic difference between a software component and an asset seems to be how they are obtained: while software components are produced from a mentality of code reuse, assets are architecture-driven. In both cases, code is the element to be reused (the architecture is never reused in the case of assets).

With regard to **software architecture/design reuse**, software architecture is one of the software elements to be reused during the software process (of course, there will be software components to support this architecture, but nothing is said in any of the approaches about how they should be built).

The ultimate step in software product reuse would be to try to bring reuse into the analysis phase of the software process, that is, **software requirements reuse**. If requirements are reused, there will be some software architecture to support these requirements, and also software components to support the architecture.

Not much research has been carried out on **software process reuse**. This kind of reuse deals with the construction of reusable software processes as a means of improving an organisation's software process.
One of the fundamental misconceptions regarding the nature of software and its creation is that principles, techniques, and tools are generally applicable, and there is no need to investigate their limits in different project contexts [Rombach92]. In [Birk97], the author recognises the need to investigate and document the application domains of software technologies, for **software technology reuse**.

Finally, there are DA methods for **software experiences reuse**. These methods try to reuse every useful experience in software systems development. It is not easy to determine experiences that are relevant to a given project context; that is, the kind of experiences that are applicable to the project.

Figure 1 shows how the different DA methods have been classified and also how the methods have been allocated to the classes. We have selected those that explicitly call themselves DA methods, as there is no agreement in the definition of what DA is.

| | | | |
|---|---|---|---|
| Reuse in Software Engineering | Software Product reuse | Software Component reuse | Draco [Neighbors84, Freeman84, Biggerstaff89]<br>Mc Cain [McCain85]<br>Prieto-Díaz [Prieto-Díaz87, Prieto-Díaz91a]<br>Simos [Simos91] |
| | | Asset reuse | HP [Cornwell96]<br>ODM [Simos95, Simos96] |
| | | Architecture/Design reuse | FODA [Kang90, Cohen92, Krut96, Schnell96]<br>IdeA [Lubars91]<br>STARS [Prieto-Díaz91]<br>DADO [Maccario97] |
| | | Requirements reuse | Synthesis [Jaworski90, Campbell90]<br>JODA [Holibaugh93] |
| | Software Process reuse | Hollenbach & Frakes [Hollenbach95, Hollenbach96] | |
| | Software Technology reuse | Birk [Birk97] | |
| | Software Experience reuse | Basili et. Al. [Basili91, Basili94]<br>Henniger [Henninger95, Henninger95a, Henninger95b, Henninger96] | |

Figure 1: Classification of Domain Analysis methods

## 3. CRITERIA FOR THE EVALUATION OF DA METHODS

Having classified the DA methods, the next step is comparison. The goal of the comparison is to discover the main differences between the DA methods and to find out why this wide variety of methods has emerged: whether it is because the methods are over-specific; because they do not output the expected results, etc. The specified comparison criteria should answer the following question: which parameters define a DA method? The initially proposed parameters are:

1. DA goal (or reuse goal).
2. Definition of the term DA.
3. Concept of domain.
4. Process for obtaining the domain model (which finally will not be taken into account).
5. DA techniques, for: analysis, classification and representation of variability.

Although they all use DA as method, all the surveyed approaches seek to solve a different problem; that is, the **DA goal** is different. All the problems have in common that they fall within the reuse framework, but they vary, some of them focusing on the development of libraries of reusable software elements, others on the improvement of the reusable elements, others on the automation of the process for producing the reusable elements, etc. Table 1 shows the objectives of the different DA methods.

| | |
|---|---|
| DA goal | Improve reusable elements |
| | Populate libraries of reusable elements |
| | Integrate DA into the software process |
| | Decrease adaptation costs |
| | Construct reusable elements |

Table 1: Purpose of DA methods.

There is no agreement (although it may appear otherwise) on the **definition of the term DA**. As shows Table 2, each author's definition is strongly influenced by the expected outcome of the DA process or by the process itself (this should not occur). There are some methods which prefer not to use the DA concept, as they think it is an ambiguous term due to the host of meanings attributed to it (as in ODM [Simos95]).

| | |
|---|---|
| Definition of DA | Identify objects and operations of a class of systems |
| | Determine characteristics that satisfy the optimum domain |
| | Activity that precedes system analysis |
| | Specify the domain model of a library |
| | Characterise a software domain to support reuse |
| | Process of identifying, organising and representing the relevant information of a domain |
| | Systems analysis for a set of systems |
| | Process where user/client knowledge is identified, elaborated and organised |
| | Identify, organise and model information to produce software requirements |
| | Process in which a reusable software architecture and reusable code are defined |
| | Identify and organise knowledge of a type of problems to describe and solve them |
| | Identify domains in which reuse of certain experiences is effective |
| | Identify patterns which are repeated throughout different development projects |

Table 2: Definitions of the term DA.

Traditionally, the **concept of domain** has represented the type of problem to be solved, that is, the domain characterised the type of application for which reusable elements were built (embedded software for satellites, avionics software, etc.). Since DA methods started to be applied to output reusable artefacts other than software products, the concept of domain has changed, and is now the context in which the software elements can be used or the type of software project (project characteristics). Table 2 shows all the possible values for this parameter.

| | |
|---|---|
| Domain | Functionality |
| | Application context |
| | Type of software project |

Table 3: Domain concepts.

All these DA methods may appear at first glance to follow a different **process for obtaining the domain model**, but the comparative survey performed by Arango [Arango94] showed that all DA methods follow what he calls a *common process*, shown in Figure 2. Arango defines this common process as follows:

- *Domain characterisation and project planning:* This is a feasibility analysis and planning phase.
- *Data collection*: Phase driven by the modelling needs. The data collection sources may vary from experts to documents, etc.
- *Data analysis (or subsidiary modelling)*: The purpose of this phase is to build descriptions of reusable elements, identifying the similarities and differences between them.
- *Classification (or primary modelling)*: The information modelled in the previous step is refined, clustering together similar descriptions, abstracting relevant common features from the descriptions in each cluster, adding new descriptions to existing or to new clusters, or reorganising the clusters. Abstractions are organised into generalisation hierarchies.
- *Evaluation of the domain model*: This activity evaluates the domain model obtained, and any defects found are put right. Some methods include references to evaluation steps in their process, though they fail to explicitly specify tasks or performance objectives for the models.

Therefore, although it initially looked interesting, **this parameter will not be taken into account** for comparing different methods, as they all follow a similar process and, accordingly, this parameter does not discriminate.

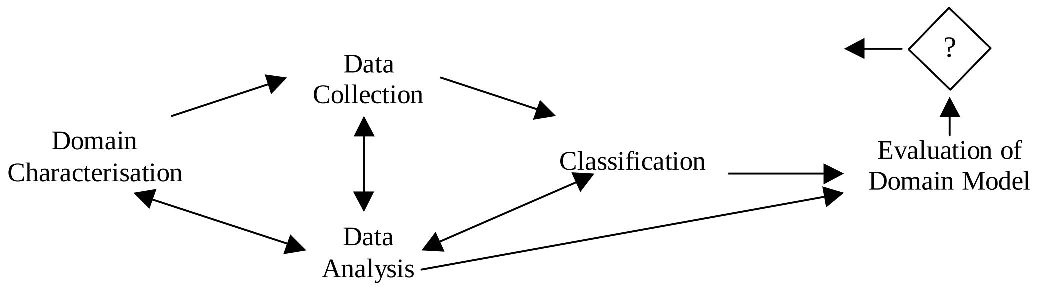

Figure 2: Representation of the Common Process followed by all DA methods

The above-mentioned survey by Arango also points out that not all the techniques used in DA are **DA techniques**. As a result, only the following used techniques will be of any use for the purpose of this paper:

- **Analysis technique***:* Different techniques can be used for data analysis during subsidiary modelling, are shown in Table 3.

| | |
|---|---|
| Analysis technique | Object oriented |
| | Functional (and data) decomposition |
| | Qualitative analysis |
| | Quantitative analysis |
| | Case-based technology |

Table 4: Data analysis techniques.

- **Classification technique**: There are a wide variety of techniques used in primary modelling. These are shown in Table 4.

| | |
|---|---|
| Classification | Facets |
| | Features |
| | Capabilities |
| | Requirements |
| | 3C´s model |
| | Coad-Yourdon |
| | Partial patterns |

Table 5: Classification techniques.

- **Representation of variability***:* Not all the methods represent variability in the domain in the same way. Some methods do not consider variations (or are assumed no to because this is not mentioned). All the different techniques identified for representing variability in the domain are set out in Table 5.

| | |
|---|---|
| Representation of variability | Combinations |
| | Compromises |
| | Specialisation |
| | Refinements |
| | Relations |
| | Parameterisation |
| | Code generation |

Table 6: Techniques used to represent variability in the domain.

## 4. RESULTS OF THE EVALUATION

Here, the results of the evaluation (according to the parameters explained in the previous section) are shown. The results appear in the form of tables, each representing a different group in the classification.

Thus, Table 7, Table 8, Table 9 and Table 10 show the results obtained when comparing DA methods for the construction of reusable products (software components, assets, software architectures and software requirements respectively). Table 11 sets out the results obtained when comparing different DA methods for the construction of reusable processes. Table 12 shows the results obtained when comparing DA methods for the construction of reusable technologies, and finally, Table 13 sets out the results obtained when comparing different DA methods for the construction of and reusable experiences.

| DA METHOD | DA GOAL | DA DEFINITION | DOMAIN | DA TECHNIQUES | | |
|---|---|---|---|---|---|---|
| | | | | ANALYSIS TECHNIQUE | CLASSIFICATION TECHNIQUE | REPRESENTATION OF VARIABILITY |
| Neighbors [Neighbors84,] | Improve reusable elements | Identify objects and operations of a class of systems | Functionality | -- | -- | Code generation |
| Mc Cain [McCain85] | Decrease adaptation costs | Determine characteristics that satisfy optimum domain | Functionality | Functional decomposition | -- | Compromises |
| Prieto-Díaz [Prieto-Díaz87] | Improve reusable elements | Activity that precedes system analysis | Functionality | Functional decomposition | Facets | Combinations |
| Simos [Simos91] | Populate libraries of reusable elements | Specify the domain model of a library | Functionality | Functional decomposition | Features | Parameterisation<br>Code generation<br>Combinations |

Table 7: Comparison of DA methods for the construction of reusable components

| DA METHOD | DA GOAL | DA DEFINITION | DOMAIN | DA TECHNIQUES | | |
|---|---|---|---|---|---|---|
| | | | | ANALYSIS TECHNIQUE | CLASSIFICATION TECHNIQUE | REPRESENTATION OF VARIABILITY |
| HP [Cornwell96] | Construct reusable elements | Characterise a software domain to support reuse | Functionality | Functional decomposition | Capabilities | Relations |
| ODM [Simos95, Simos96] | Construct reusable elements | -- | Functionality | -- | Features | Combinations |

Table 8: Comparison of DA methods for the construction of reusable assets

| DA METHOD | DA GOAL | DA DEFINITION | DOMAIN | DA TECHNIQUES | | |
|---|---|---|---|---|---|---|
| | | | | ANALYSIS TECHNIQUE | CLASSIFICATION TECHNIQUE | REPRESENTATION OF VARIABILITY |
| FODA [Kang90,Cohen92, Krut96,Schnell96] | Construct reusable elements | Process of identifying, organising and representing the relevant information of a domain | Functionality | Functional decomposition | Features | Refinements |
| IDeA [Lubars91] | Populate libraries of reusable elements | -- | Functionality | Functional decomposition | -- | Specialisation and Refinements |
| STARS [Prieto-Díaz91] | Populate libraries of reusable elements | Systems analysis for a set of systems | Functionality | Functional decomposition | Facets | Not clear |
| DADO [Maccario97] | Integrate DA into the software process | Process where user/client knowledge is identified, elaborated and organised | Functionality | Object Oriented | Features | Not stated |

Table 9: Comparison of DA methods for the construction of reusable architectures

| DA METHOD | DA GOAL | DA DEFINITION | DOMAIN | DA TECHNIQUES | | |
|---|---|---|---|---|---|---|
| | | | | ANALYSIS TECHNIQUE | CLASSIFICATION TECHNIQUE | REPRESENTATION OF VARIABILITY |
| Synthesis [Jaworski90, Campbell90] | Decrease adaptation costs | Identify, organise and model information to produce software requirements | Functionality | Object oriented | Requirements | Parameterisation |
| JODA [Holibaugh93] | Construct reusable elements | Process in which a reusable software architecture and reusable code are defined | Functionality | Object oriented | Coad-Yourdon | -- |

Table 10: Comparison of DA methods for the construction of reusable requirements

| DA METHOD | DA GOAL | DA DEFINITION | DOMAIN | DA TECHNIQUES | | |
|---|---|---|---|---|---|---|
| | | | | ANALYSIS TECHNIQUE | CLASSIFICATION TECHNIQUE | REPRESENTATION OF VARIABILITY |
| Hollenbach & Frakes [Hollenbach95, 96] | Populate libraries of reusable elements | -- | Application context | Functional decomposition | 3C´s model | Compromises |

Table 11: Comparison of DA methods for the construction of reusable processes.

| DA METHOD | DA GOAL | DA DEFINITION | DOMAIN | DA TECHNIQUES | | |
|---|---|---|---|---|---|---|
| | | | | ANALYSIS TECHNIQUE | CLASSIFICATION TECHNIQUE | REPRESENTATION OF VARIABILITY |
| Birk [Birk97] | Construct reusable elements | Identify and organise knowledge of a type of problems to describe and solve them | Type of software project | Qualitative analysis | Facets | -- |

Table 12: Comparison of DA methods for the construction of reusable technology

| DA METHOD | DA GOAL | DA DEFINITION | DOMAIN | DA TECHNIQUES | | |
|---|---|---|---|---|---|---|
| | | | | ANALYSIS TECHNIQUE | CLASSIFICATION TECHNIQUE | REPRESENTATION OF VARIABILITY |
| Basili [Basili91,i94] | Construct reusable elements | Identify domains in which reuse of certain experiences is effective | Type of software project | Qualitative analysis Quantitative analysis | Features | -- |
| Henninger [Henninger95, 95a,95b,96] | Populate libraries of reusable elements | Identify patterns which are repeated throughout different development projects | Application context | Case-based technology | Partial patterns | -- |

Table 13: Comparison of DA methods for the construction of reusable experiences

In every table, the following information is shown: The first column shows the goal of the DA, with values taken from Table 1. The second column shows the definition that each method gives of the term DA, with values taken from Table 2. The third column shows the concept of what a domain is, with values taken from Table 3. Columns 4 to 6 show the different techniques each method uses for performing DA: Column 4 shows the analysis techniques (taken from Table 4). Column 5 shows the classification techniques (taken from Table 5), and finally, column 6 shows techniques used for representing the variability in the domain (taken from Table 6).

## 5. CONCLUSIONS

The definition of what DA is, its objective and the concept of domain, is strongly influenced by the type of element to be reused. But reuse needs change as time goes by, and elements other than software products should be reused. This is the reason for the emergence of new DA methods. Unfortunately, analysis techniques used in DA are dependent on the type of element to be reused. While the methods that are intended for software product reuse use analysis techniques derived from Software Engineering, DA methods that are intended for the reuse of other software elements use alternative techniques. Thus, we can conclude that the analysis techniques used in current DA methods are not generic enough.

So, there is a need for DA methods that can be used to build any type of software element, as called for by current reuse needs (and it does not matter whether or not the technique is generic or dependent from the type of element to be reused). Also, new modelling techniques are needed, to allow an adequate characterisation of every kind of reusable element and not only software products.

## 6. REFERENCES

[Arango91] G. Arango and R. Prieto-Díaz. Domain analysis concepts and research directions. *In Domain Analysis and Software Systems Modelling*, pages 9-33, Washington DC, 1991. IEEE Computer Society.

[Arango94] G. Arango. *Software Reusability, chapter 2.* Domain analysis methods, pages 17-49. Workshops M.E. Horwood, London 1994.

[Basili91] V.R. Basili and H.D. Rombach. Support for comprehensive reuse. *Software Engineering Journal,* pages 303-316, September 1991.

[Basili92] V.R. Basili and G. Caldiera. A reference architecture for the component factory. *ACM Transactions on Software Engineering and Methodology*, Vol.1 No. 1, January 1992, pages 53-80.

[Basili94] V.R. Basili, L.C. Briand and W.M. Thomas. Domain analysis for the reuse of software development experiences. *In Proceedings of the 19th Annual Software Engineering Workshop, NASA/GSFC*, pages 11-24. NASA/GSFC. December 1994.

[Biggerstaff89] T. Biggerstaff and A.J. Perlis. *Software Reusability, vol. 1, chapter 12.* Draco: A method for engineering reusable software systems, pages 295-320. Addison-Wesley, 1989.

[Birk97] A. Birk. A knowledge acquisition method for domain analysis of software engineering technologies. *IESE-Report 019.97*, Fraunhofer IESE, 1997.

[Campbell90] G.H. Campbell Jr., S.R. Faulk and D.M. Weiss. Introduction ot synthesis. *Interim Report Intro-Synthesis-90010-N. Version 01.00.01*, Software Productivity Consortium, 2214 Rock Hill Road. Herndon, Virginia 22070, June 1990.

[Cohen92] S.H. Cohen, J.L. Stanley Jr., A.S. Peterson and W.R. Krut Jr. Application of feature-oriented domain analysis to the army movement control domain. *Technical Report CMU/SEI-91-TR-28*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, June 1992.

[Cornwell96]P.C. Cornwell. HP domain analysis: Producing useful models for reusable software. *Hewlett-Packard Journal*, article 5. August 1996.

[Freeman84] P. Freeman. Analysis of the draco approach to constructing software systems. *Technical Report 85-07*, Information and Computer Science, University of California, Irvine, Ca 92717. October 1984.

[Henninger95] S. Henninger, K. Lappala and A. Raghavendran. An organizational learning approach to domain analysis. In *Proceddings of the 17th International Conference on Software Engineering ICSE-17*, pages 95-104, Seattle, Washington, 1995. ACM Press.

[Henninger95a] S. Henninger. Accelerating successful reuse through the domain lifecycle. In *Proceedings of the Seventh Workshop on Institutionalizing Software Reuse*. St. Charles, Illinois. August 1995. Andersen Consulting Center.

[Henninger95b] S. Henninger. Developing domain knowledge through the reuse of project experiences. In Mansur Samadzadeh and Mansour Zand, ediors, *Proceedings of the Symposium on Software Reusability, SSR'95*, pages 186-195. Seattle, Washington. 28-30 April. ACM Press.

[Henninger96] S. Henninger. Accelerating the successful reuse of problem solving knowledge through the domain lifecycle. *In Proceedings of the Fourth International Conference on Software Reuse*, pages 124-133, Orlando, Florida, April 1996.

[Holibaugh93] R. Holibaugh. Joint integrated avionics working group (JIAWG) object-oriented domain analysis method (JODA). Version 3.1. *Special Report CMU/SEI-92-SR-3*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213. November 1993.

[Hollenbach95] C. Hollenbach and W. Frakes. Software process reuse. *In Proceedings of the Seventh Workshop on Institutionalizing Software Reuse*, St. Charles, Illinois, August 1995. Andersen Consulting Center.

[Hollenbach96] C. Hollenbach and W. Frakes. Software process reuse in an industrial setting. *Technical Report TR-96-04*, Department of Computer Science, Virginia Tech., 2990 Telestar CT., Falls Church, VA 22042, January 1996.

[Jaworski90] A. Jaworski, F. Hills, T. Durek, S. Faulk and J. Gaffney. A domain analysis process. *Interim Report-Domain Analysis Project Domain-Analysis-90001-N*. Version 01.00.03, Software Productivity Consortium, 2214 Rock Hill Road. Herndon, virginia 22070, January 1990.

[Kang90] K.C. Kang, S.H. Cohen, J.A. Hess, W.E. Novak and A.S Peterson. Feature-oriented domain analysis feasibility study: interim report. *Technical Report CMU/SEI-90-TR-21*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, August 1990.

[Krut96] R. Krut and N. Zalman. Domain analysis workshop report for the automated prompt and response system domain. *Technical Report CMU/SEI-96-SR-001*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, May 1996.

[Leach97] R.J. Leach. *Software Reuse*. Mc. Graw-Hill, New York, 1997.

[Lubars91] M.D. Lubars. Domain analysis and domain engineering in IDeA. *IEEE Tutorial: Domain Analysis and Software Systems Modeling*. Pages 163-178. IEEE Computer Society Press, Los Alamitos, California, 1991.

[Maccario97] P.M. Maccario. The domain analysis integrated in an object oriented development methodology. *In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse*, Columbus, Ohio, March 1997. Ohio State.

[McCain85] R. McCain. Reusable software component construction: A product oriented paradigm. *In Proceedings of the 5th AIAA/ACM/NASA/IEEE Computers in Aerospace Conference*, pages 125-135, 1985.

[McClure97] C. McClure. *Software Reuse Techniques. Adding Reuse to the Systems Development Process*. Prentice-Hall, New Jersey, 1997.

[Neighbors84] J. Neighbors. The draco approach to constructing software from reusable components. IEEE Transactions on Software Engineering, Vol. SE-10, no. 5, pages 564-574, September 1984.

[Prieto-Díaz87] R. Prieto-Díaz. Domain analysis for reusability. *In Proceddings COMPSAC'87*, pages 23-29, Tokio, Japan, October 1987.

[Prieto-Díaz91] R. Prieto-Díaz. Domain analysis process model, a domain analysis method developed by IBM federal sector division and available in reuse library process model. *Report IBM STARS CDRL 03041-00*2, U.S. Air Force Electronic Systems Center, Hanscom Air Force Base, MA, July 1991. Also available from ASSET, (304) 594-3954. ASSET-A-177.

[Prieto-Díaz91a] R. Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, pages 88-97. May 1991.-

[Rombach92] H.D. Rombach. Systematic software technology transfer. In V.R. Basili, H.D. Rombach and R.W. Selby editors, *Experimental Software Engineering Issues. International Workshop*, Daughstuhl Castle, Germany, September 1992.

[Rombach94] H.D. Rombach and W. Schafer. *Software Reusability, chapter 5*. Tools and environments, pages 113-152. Workshops M. E. Horwood, London 1994.

[Schnell96] K. Schnell, N. Zalman and A. Bhatt. Transitioning domain analysis: An industry experience. Technical Report CMU/SEI-96-TR-009, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15214. June 1996.

[Simos91] M.A. Simos. The growing of an organon: A hybrid knowledge-based technology and methodology for software reuse, pages 204-221. *IEEE Tutorial: Domain Analysis and Software Systems Modelling*. IEEE Computer Society Press, Los Alamitos, California, 1991.

[Simos95] M.A. Simos. Organization domain modeling (ODM): formalizing the core domain modeling life cycle. In Mansur Samadzadeh and Mansour Zand, editors, *Proceedings of the Symposium on Software Reusability*. SSR '95, pages 196-205, Seattle, Washington, 28-30 April 1995. ACM Press.

[Simos96] M.A. Simos, D. Creps, C. Kingler, L. Levine and D. Allemang. Organization domain modeling (ODM) guidebook. Version 2.0. *Informal Technical Report STARS-VC-A025/001/00*, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road. Manassas, VA 22110-4121. June 1996.

[Wartik92] S. Wartik and R. Prieto-Díaz. Criteria for comparing reuse-oriented domain analysis approaches. International Journal of Software Engineering and Knowledge Engineering, vol. 2, no. 3, pp. 403-431. September 1992