Journal of Universal Computer Science, vol. 9, no. 7 (2003), 659-669
submitted: 3/2/03, accepted: 21/7/03, appeared: 28/7/03 © J.UCS

Experience Base Schema Building Blocks of the PLEASERS Library

Raimund L. Feldmann
(University of Kaiserslautern, Germany
r.feldmann@computer.org)

Ralf Carbon
(University of Kaiserslautern, Germany
carbon@informatik.uni-kl.de)

Abstract: Quality and process improvement programs usually require organizations to run a repository such as an experience base. However, setting up the schema of an experience base requires expert knowledge. But schema experts are not always available to support the setup of a new experience base. One promising solution is to capture their knowledge in patterns or building blocks. An initial collection of such building blocks is systematically documented in the PLEASERS (Product Line Approach for Software Engineering Repositories) library. In this article we describe the underlying conceptual model of the PLEASERS schema building blocks. Schema experts can use the introduced model to create sets of schema building blocks representing their knowledge.

Keywords: Experience Base Schemata, Schema Building Blocks, Repository Schema Reuse, Knowledge & Experience Management
Categories: H.2.1, D.2.2, D.2.11, D.2.13, H.2.3

1 Motivation

Organizational learning –often based on quality and process improvement programs– usually focuses on experience gained in past projects. Thus, reusing successfully applied (code) components and other means of knowledge is widely accepted in research and industry. As a result, patterns and frameworks [Gamma, 95], for instance, are being developed to capture the gained experience of software that has already been developed. To support the underlying idea of comprehensive reuse (e.g., [Basili, 91]), repositories are usually installed and operated as a core system of a Learning Software Organization [Bomarius, 98]. Such a software engineering repository (SE Repository) is used for storing the knowledge and experience of an organization, and providing it to new projects upon request. The often applied Experience Factory concept by Basili et. al. [Basili, 94] suggests the implementation of a comprehensive organizational SE Repository denoted as Experience Base (EB). Publications on how to (technically) install an EB do exist (e.g., [Basili, 91], [Tautz, 99], [Broomé, 00]), as do publications discussing the challenges and pitfalls in designing and tailoring an EB for organizational needs (e.g., [Koennecker, 99], [Lindvall, 01], [Schneider, 02]). However, setting up a suitable schema for a new EB remains a difficult and arduous task that requires expert knowledge. But schema experts are not always available for a company to support the setup of a new EB with their experience.

Let us consider the following situation: A company, let us say ITS+M (*IT Solutions + More)*, wants to install an EB as part of their improvement program. Our company primarily does consulting for small and medium-sized enterprises that need to optimize their software processes. Therefore, the new EB should systematically store process patterns (e.g., [Gnatz, 99]) that are often employed by ITS+M to optimize their customers' SW processes. ITS+M has never run an EB before, and does not employ an expert who knows how to implement and run such an EB.

For ITS+M it would be helpful if they were able to access an archive with standardized EB schema elements –similar to their own process patterns– that represent schema expert knowledge on how to store process models or process patterns in an EB. As part of the Product Line Approach for Software Engineering Repositories (PLEASERS), such a schema library has been developed. So-called schema building blocks (schema BBs) are used for documenting and consolidating the schema knowledge in the PLEASERS library. The conceptual model for these schema building blocks is detailed in this article. Schema experts can thereby record their knowledge and provide (i.e., transfer) it to organizations that are currently building up their own EB, without being present in person.

The remainder of this article is organized as follows: [Section 2] describes the context in which the schema BBs are used. Next, the structure of our schema BBs is introduced [section 3]. Different types of schema BBs are distinguished. Then, in [section 4], we give an example of how the introduced schema BB types can be used for capturing schema expert knowledge. A tool environment supporting the creation of sets of schema BBs in accordance with our model is presented in [section 5]. Finally, we summarize our results and conclude with future directions in [section 6].

2 PLEASERS and the PLEASERS Schema Library

Currently, PLEASERS is being developed at the University of Kaiserslautern. PLEASERS supports the reuse-based development of schemata for *new* SE Repository systems without having an expert at hand. The approach is based on the product line idea [Weiss, 99]. From a predefined scope, users can precisely characterize the SE Repository to be build (i.e., fix the repository requirements), and, thereby, retrieve documented solutions from schema experts stored in the PLEASERS library.

Fixing the requirements of the new SE Repository schema is achieved by answering a questionnaire. The questionnaire can be compared to the decision model [Clements, 01] in product lines. With the given answers one can automatically select the appropriate solutions of schema experts from the PLEASERS library. The PLEASERS library itself stores best practices and well-tried schema solutions in the form of the schema building blocks described in the next [section 3].

Based on the idea of a modular SE Repository structure [Feldmann, 00], PLEASERS suggest an initial set of schema BBs for constructing new EB schemata with PLEASERS. The initial set of schema BBs was derived from the author's own

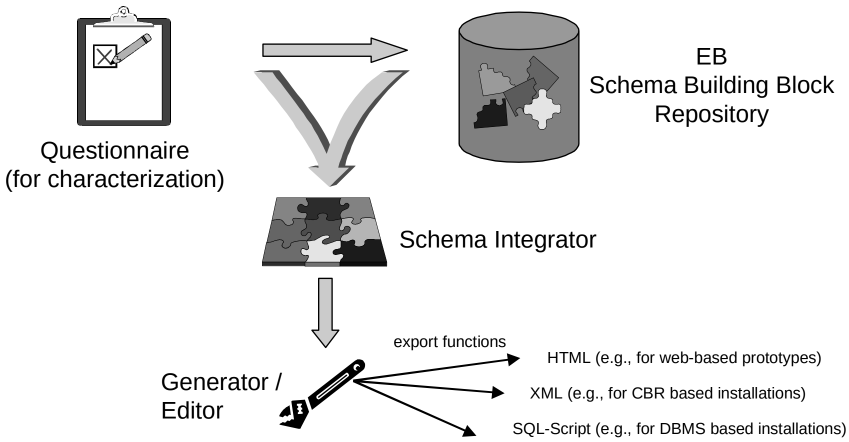

Figure 1: Outline of the PLEASERS tool suite

experience with developing the SFB-EB, the comprehensive Experience Base of the *Sonderforschungsbereich 501* (SFB 501) [Avenhaus, 98].

PLEASERS is supported by the tool suite illustrated in [Fig. 1]. The tool suite automates the PLEASERS process to a large extent. Users applying PLEASERS must first fill in a simple-to-use electronic questionnaire [Trapp, 02]. These questions are stored together with each BB. Based on the answers provided, applicable schema BBs are selected from the PLEASERS library. A schema integrator then automatically combines the retrieved schema BBs and displays the constructed schema in a graphical editor. With the help of this editor, users can adapt and tailor the constructed PLEASERS schema to their specific needs. Since the whole process employs no specific technology, schemata developed with PLEASERS and the PLEASERS tool suite are technology independent. Only at the very end of the development process, the user chooses a certain export function to generate an instance of the developed schema. This export function then uses a specific technology depending on the intended implementation platforms for the new SE Repository system. Therefore, the final schema is generated in the form of SQL-Scripts, XML descriptions, or HTML representations.

After this brief introduction of the context, we will now focus on the conceptual model of the PLEASERS schema BBs.

3 A Conceptual Model for Schema Building Blocks

Different types of content are stored in an EB. According to [Aamodt, 95] these are data (e.g., measurement data), information (e.g., effort distribution models), and knowledge (e.g., process patterns). For an EB we add experience (e.g., lessons learned in a specific project) as a fourth type. The schema of an EB must support the storage of all four types of content. Consequently, schema BBs must capture structures for different EB entries.

Our conceptual model [Carbon, 02] describes such schema BBs and classifies them in a UML-like notation as illustrated in [Fig. 2]. A schema BB captures all information regarding attributes, relations, and constraints, that is necessary to describe an existing solution, and needed for generating a new schema by reusing these information in another context. Currently, we distinguish three types of schema BBs: *Root Building Block (RBB)*, *Element Specific Root Building Block (ESRBB)*, and *Add On Building Block (Add On)*. Components common to all BB types are attributes, relations, and constraints.

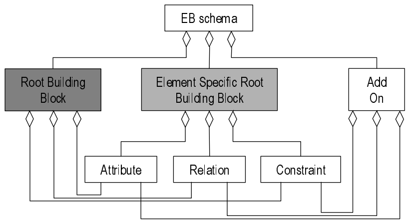

Figure 2: Conceptual schema building block model

- An RBB encapsulates common attributes and relations applicable to all entries in an EB, regardless of their content type. Examples for attributes of an RBB are: "`name`", "`creation_date`", or "`short_description`"; a possible relation of an RBB could be "`also_known_as`". RBB attributes and relations can be regarded as a basis for storing all kinds of data, information, knowledge, or experience in an EB. Once defined, a RBB can be reused in all new EB schemas.
- An ESRBB contains attributes and relations necessary to represent characteristics of specific EB entries. ESRBBs can be compared to classes that help in structuring and categorizing the EB schema. An initial set of ESRBBs can be defined by studying the modular repository structure found in [Feldmann, 00]. This leads, for example, to ESRBBs for process patterns (e.g., "ProcessPattern" in [Fig. 4]) or lessons learned (e.g., "LessonLearned" in [Fig. 4]). Attributes specific to a process pattern could be "`application_domain`" and "`lifecycle_model`"; a possible relation is "`see_also`", which allows pointers to similar process patterns. The ESRBB for lessons learned could consist of the attributes "`situation`", "`problem`", and "`solution`". These examples illustrate that ESRBBs can contain attributes and relations to store context information. The storage of such context information is required, in particular, for the content types knowledge and experience. According to [Basili, 91], such context attributes are essential for identifying adequate reuse candidates in an EB. But an ESRBB can also be used to store EB entries of the content type information. Such an ESRBB then only contains attributes and relations to represent context independent information. An ESRBB "Person", for instance, could capture information about employees (e.g., with the attributes "`employee_name`" and "`phone#`").

- An Add On, being the third type of schema BBs in our conceptual model, allows the flexible adaptation of EB schemas to special requirements. Let us suppose a schema expert wants to express that for some organizations, it may be useful to store ownership information for EB entries (e.g., process pattern) in the schema. An "owns/owned_by" relation between the ESRBB "Person" and the corresponding ESRBB "ProcessPattern" could model this. However, by simply adding an "owns/owned_by" relation to the ESRBBs, this relation would always be included in all schemas that make use of these ESRBBs. To avoid such problems in our conceptual model, a schema expert would use an Add On "OwnedBy". This Add On would capture the relation to the ESRBB "Person" and would only be selected if ownership needs to be documented in the EB schema. Note that an Add On cannot stand alone. It always depends on the definition of at least one ESRBB.

Dependencies between schema BBs (i.e., the RBB, ESRBBs, and Add Ons) are specified in our conceptual model by constraints. They guarantee the correct composition of BBs to create an EB schema. Our constraints for schema BBs are described by the grammar, in a BNF-like notation. The set of non-terminal symbols is declared as {C, B, B_{list}, D_{BB}}, where "C" is the start symbol. The set of terminal symbols is declared as {<building block>, **<extended by>**, **<requires>**, **<mutual exclusive>**, **<and>**}, where dependencies are highlighted in **bold** and <building block> stands for a single BB from the set of existing BBs. The set of productions is depicted in [Fig. 3].

{ 1) C $\rightarrow$ $BD_{BB}B_{list}$
2) B_{list} $\rightarrow$ B | B **<and>** B_{list}
3) D_{BB} $\rightarrow$ **<extended by>** | **<requires>** | **<mutual exclusive>**
4) B $\rightarrow$ <building block>
}.

Figure 3: Grammar for schema BB constraints

Our Add On "OwnedBy", for instance, requires two ESRBBs: the "ProcessPattern" ESRBB and the "Person" ESRBB. This dependency is specified in the form of a constraint as:

"OwnedBy" **<requires>** "ProcessPattern" **<and>** "Person".

An additional constraint

"OwnedBy" **<requires>** "LessonLearned" **<and>** "Person"

indicates that the same Add On could also be used to instantiate an ownership relation for lessons learned in an EB schema. From this example it becomes obvious that a single Add On can easily be combined with many ESRBBs. For a more detailed discussion of dependencies between different types of BBs and their representation with the help of constraints, the interested reader is referred to [Carbon, 02].

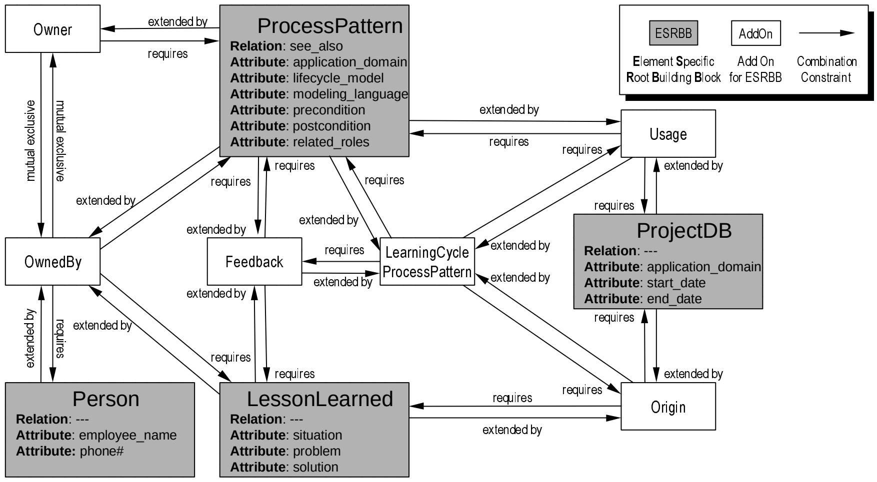

Figure 4: ESRBBs and Add Ons of our example with their constraints

4 Capturing Schema Expert Knowledge Using our Model

Now we will have a closer look at our example set of BBs depicted in [Fig. 4]. Each BB, especially the ESRBBs and Add Ons, can be independently defined by an expert for the corresponding type of schema.

Let us assume that in addition to the attributes already mentioned in [section 3], a process pattern schema expert completed the "ProcessPattern" ESRBB by adding the attributes "modeling_language", "precondition", "postcondition", and "related_roles". Furthermore, the schema expert documents that it should be possible to identify the owner of a stored process pattern, even if this person is not an employee. Consequently, s/he defines the Add On "Owner", which contains an attribute "owner_name" for storing the ownership information. The Add On needs the ESRBB "ProcessPattern" to be applicable. This is expressed by another "*requires*" constraint in [Fig. 4]. Now the ESRBB "ProcessPattern" requires either "Owner" or "OwnedBy" for identification of the ownership of a process pattern. Since the application of both Add Ons, "Owner" and "OwnedBy", in the same EB schema would lead to redundancy (which again might lead to inconsistencies later in the EB content) both Add Ons are declared as *"mutual exclusive*" by using another constraint.

Let us further assume that we asked an expert for Learning Software Organizations to help us in completing our set of schema BBs. According to this expert, it should be possible to document learning cycles in an EB. Therefore, s/he enriches our example set with the following BBs:

The ESRBB "ProjectDB" allows the storage of project information as a basis for organizational learning. This ESRBB holds attributes such as "application_domain", "start_date", and "end_date" of the project.

The Add On "Usage" defines a relation "used_in/uses" between the ESRBBs "ProjectDB" and "ProcessPattern" to indicate that a process pattern of the EB has been used in a certain project. Constraints indicate that the Add On requires both ESRBBs to exist before it can be integrated into an EB schema.

The Add On "Origin" holds the definition for a relation "`gained_in/gains`" between the ESRBBs "ProjectDB" and "LessonLearned". It allows to indicate from which project of the EB a lesson learned was derived. Again, the "*requires*" and "*extended by*" constraints are used to express the dependencies in combining the Add On and ESRBBs.

The Add On "Feedback" allows a relation "`has_part/is_about`" between the ESRBBs "ProcessPattern" and "LessonLearned" in the EB. Hence, one can store feedback in the form of lessons learned for a concrete process pattern in the EB.

The Add On "LearningCycle_ProcessPattern" allows the definition of a learning cycle for process patterns based on feedback gained in concrete projects. To install the learning cycle, this Add On simply requires the usage of the Add Ons "Feedback", "Usage", and "Origin". This is coded with the help of a set of "*requires*" constraints. Consequently, the Add On does not contain any specific attributes or relations.

This should close the integration of expert knowledge into our example set of schema BBs. The given set already allows us to support the creation of EB schemas with up to four different types of entries. All of these possible entries are basically described by the four ESRBBs "ProcessPattern", "LessonLearned", "ProjectDB", and "Person". Of course, there could be further extensions of the BBs with additional (schema) experts, but this is beyond the scope of this paper.

Now let us see how our example set of schema BBs can help our company ITS+M [see section 1] with their problem in installing a new EB. Some possible EB schemas based on combinations of our BBs are illustrated in [Fig. 5]. Since ITS+M wants to store process patterns in the new EB, all schemas initially include the ESRBB "ProcessPattern". Thereby, ITS+M already receives a schema that holds an initial set of attributes used for storing process patterns. The schema includes context attributes (e.g., "`application_domain`" and "`precondition`") that will help ITM+S to identify possible process patterns that can be used for optimizing the SW processes of a certain customer. A complete EB schema for ITS+M derived from the BB set might be:

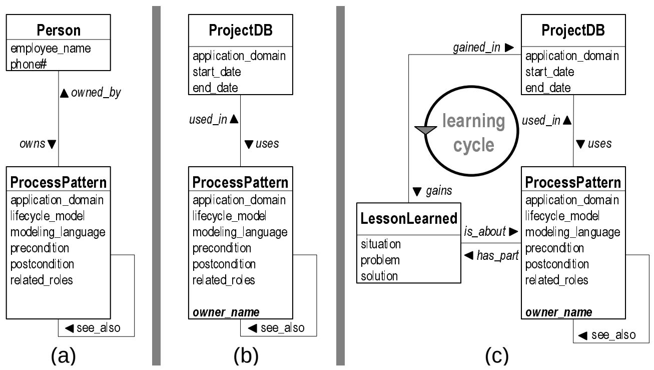

Figure 5: Example EB schema built from the set of building blocks (schema elements caused by Add On are indicated in italics)

- **Schema (a) in [Fig. 5].** This is the result of combining the ESRBBs "Person" and "ProcessPattern" with the Add On "OwnedBy". This simple schema would allow ITS+M to store their process patterns and indicate which employee can be contacted (e.g., via the phone number stored in the attribute "phone#") if questions arise. ITM+S likes the idea of documenting the owner of a process pattern. However, this solution is not selected because ITM+S does not like the idea of storing complete records with information on their employees in the new EB. Instead, ITS+M decides to use the Add On "Owner" for their schema.
- **Schema (b) in [Fig. 5].** This is the result of employing the ESRBBs "ProcessPattern" and "ProjectDB" together with the Add Ons "Owner" and "Usage". This schema is the one ITM+S favors for the initial implementation of their new EB. It allows them not only to easily select process patterns for new projects, but also to see in which similar projects a process pattern has been successfully used before.
- **Schema (c) in [Fig. 5].** Combining the ESRBB "ProcessPattern" with the Add Ons "Owner" and "LearningCycle_ProcessPattern" could build this schema. Via its required constraints, the latter includes the Add Ons "Feedback", "Usage", and "Origin". These again require the usage of the ESRBBs "ProjectDB" and "LessonLearned". As can be seen, this schema includes all elements of schema (b), and therefore, can be seen as its (modular) extension. However, since ITM+S wants to have an operable EB as soon as possible, they decide to first implement a smaller version of their EB. After this first iteration is installed successfully, they will then implement the complete schema (c) in a subsequent step.

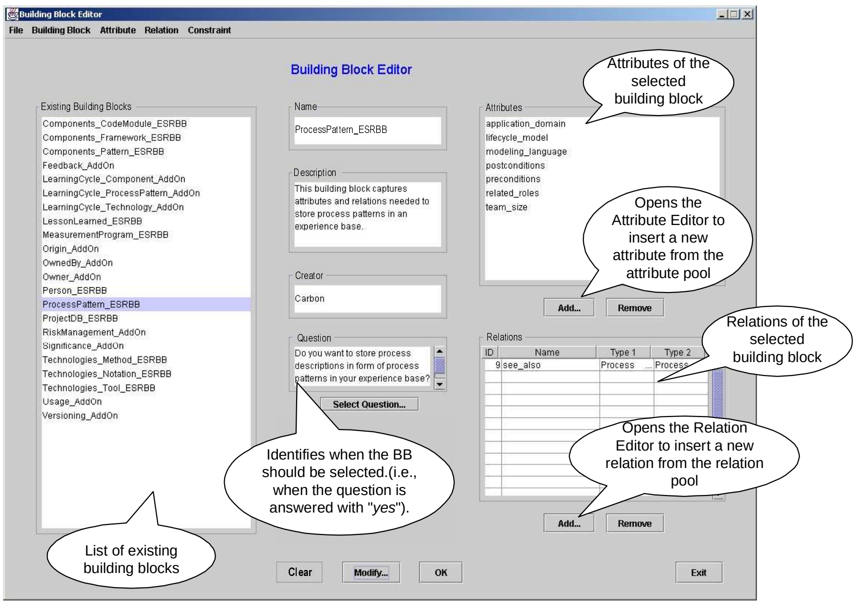

Figure 6: Screenshot of the Building Block Editor

5 Tool Support

To support easy definition and management of schema BB sets according to our conceptual model, we implemented a tool environment. The GUI is implemented in Java and a relational database management system is used to store the BB sets. Several editors are available.

An Attribute Editor and a Relation Editor provide functions to create, modify, and view attributes and relations. Attributes and relations are stored in so-called pools. When defining BBs with the Building Block Editor (see [Fig. 6]), attributes and relations are taken out of these pools. This solution supports reuse of attributes and relations in more than one BB. Furthermore, the Building Block Editor allows to store additional descriptive information with the BBs. A recommendation for selecting applicable BBs from a set, for instance, can be given. [Fig. 6] shows the Building Block Editor interface displaying the ESRBB “ProcessPattern” from our example.

The so-called Constraint Editor allows to specify dependencies between BBs of a set. Possible constraints restricted according to the productions listed in [section 3] can be easily edited without direct application of the formal productions. A complete documentation of our tool environment can be found in [Carbon, 02].

6 Conclusion and Future Directions

In conclusion we can state that the conceptual model for PLEASERS schema building blocks we presented seems to be a feasible way to document and consolidate schema knowledge. First experience in using the described approach were gained while building the underlying EB of the ViSEK portal [Visek, 03]. For this task, the initial set of schema building blocks of the PLEASERS library was used. The approach supported fast setup of an initial schema. Furthermore, it allowed focusing discussions of experts on selected parts of the schema (i.e., the schema building blocks relevant to the expert's field of knowledge). Additionally, the initial set of schema BBs in the PLEASERS library was extended after the ViSEK schema had been developed successfully. Schema BBs for storing process patterns according to the approach described in [Gnatz, 99] were added to the PLEASERS library. Hence, the knowledge of the schema experts who developed the ViSEK schema was captured and is now available to be transferred to other projects. However, the current set of schema BBs needs to be further extended to cover more areas of expertise. Additionally, the usability of the schema BB approach and the PLEASERS library, as well as its ease of use, needs to be systematically tested in future empirical studies.

Acknowledgements

Part of this work has been conducted in the context of the Sonderforschungsbereich 501 ‘Development of Large Systems with Generic Methods’ (SFB 501) funded by the Deutsche Forschungsgemeinschaft (DFG).

References

[Aamodt, 95] Aamodt, A., Nygard, M.: "Different roles and mutual dependencies of data, information and knowledge - An AI perspective on their integration"; Data and Knowledge Engineering, 16 (1995), 191–222.

[Avenhaus, 98] Avenhaus, J., Gotzhein, R., Härder, T., Litz, L., Madlener, K., Nehmer, J., Richter, M., Ritter, N., Rombach, D., Schürmann, B., Zimmermann, G.: "Entwicklung großer Systeme mit generischen Methoden - Eine Übersicht über den Sonderforschungsbereich 501"; Informatik, Forschung und Entwicklung, 13, 4 (1998), 227–234.

[Basili, 94] Basili, V.R., Caldiera, G., Rombach, D.: "Experience Factory"; In Marciniak, J.J. (ed.), Encyclopedia of Software Engineering, vol 1, John Wiley & Sons (1994), 469–476.

[Basili, 91] Basili, V.R., Rombach, H.D.: "Support for comprehensive reuse"; IEE Software Engineering Journal, 6, 5 (1991), 303–316.

[Bomarius, 98] Bomarius, F., Althoff, K.-D., Müller, W.: "Knowledge Management for Learning Software Organizations"; Software Process–Improvement and Practice, 4, 2 (1998), 89–95.

[Broomé, 00] Broomé, M., Runeson, P.: "Technical Requirements for the Implementation of an Experience Base"; In: Ruhe, G., Bomarius, F. (eds.), Learning Software Organizations: Methodology and Applications, LNCS #1756, Springer (2000), 87–102.

[Carbon, 02] Carbon, R.: "A Repository for Experience Base Schema Building Blocks", Master’s Thesis, Software Engineering Research Group, Dept. of Computer Science, University of Kaiserslautern (2002).

[Clements, 01] Clements, P.C., Northrop, L.: "Software Product Lines: Practices and Patterns"; SEI Series in Software Engineering. Addison-Wesley (2001).

[Feldmann, 00] Feldmann, R.L.: "On Developing a Repository Structure Tailored for Reuse with Improvement"; In: Ruhe, G., Bomarius, F. (eds.), Learning Software Organizations: Methodology and Applications, LNCS #1756, Springer (2000), 51–71.

[Gamma, 95] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: "Design Patterns – Elements of Reusable Object-Oriented Software"; Addison-Wesley (1995).

[Gnatz, 99] Gnatz, M., Marschall, F., Popp, G., Rausch, A. Scherin, W.: "Modular Process Patterns supporting an Evolutionary Software Development Process"; Proc. 3rd International Conference on Product Focused Software Process Imrovement (PROFES 2001), Kaiserslautern, Germany (2001).

[Koennecker, 99] Koennecker, A., Jeffery, R., Low, G.: "Lessons Learned from the Failure of an Experience Base Initiative Using Bottom-up Development Paradigm"; Proc. 24th Annual Software Engineering Workshop (SWE24), Greenbelt, Maryland, USA (1999), Online @ http://sel.gsfc.nasa.gov/website/sew/1999/program.html, last visited January 2003.

[Lindvall, 01] Lindvall, M., Frey, M., Costa, P., Tesoriero, R.: "Lessons Learned about Structuring and Describing Experience for Three Experience Bases"; In: Althoff, K.-D. et. al. (eds.), Advances in Learning Software Organizations, LNCS #2176, Springer (2001), 106–119.

[Schneider, 02] Schneider, K., Schwinn, T.: "Maturing Experience Base Concepts at DaimlerChrysler"; Software Process Improvement and Practice, 6, 2 (2001), 85–96.

[Tautz, 99] Tautz, C., Gresse von Wangenheim, C.: "REFSENO: A Representation Formalism for Software Engineering Ontologies"; Proc. 5th German Conference on Knowledge-based Systems (1999).

[Trapp, 02] Trapp, M.: "A Flexible Approach for Coupling Experience Base Requirements and Applicable Schema Building Blocks"; Master's Thesis, Software Engineering Research Group, Dept. of Computer Science, University of Kaiserslautern (2002).

[Visek, 03] "ViSEK: Virtuelles Software Engineering Kompetenzzentrum"; Online @ http://visek.de, last visited January 2003.

[Weiss, 99] Weiss, D., Lai, C.T.R.: "Software Product-Line Engineering – A Family-Based Software Development Process"; Addison-Wesley, (1999).