Published in the Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE ’98)

Towards Goal-Oriented Organizational Learning: Representing and Maintaining Knowledge in an Experience Base[1]

Raimund L. Feldmann, Jürgen Münch, Stefan Vorwieger

Fachbereich Informatik, Universität Kaiserslautern, Postfach 3049, 67653 Kaiserslautern, Germany
{r.feldmann, j.muench, s.vorwieger}@computer.org

Abstract. *Reusing experience in the form of processes, products, and other forms of knowledge is essential for improvement, i.e., reuse of knowledge is the basis for improvement [3]. This paper reports a case study investigating the use of a context-oriented representation of software engineering experiences. Our claim is that the joint description of experiences and their scope of validity (including a context description) may improve 1) the selection of suitable reuse candidates, 2) the adaptation of these candidates to current project goals and characteristics, and 3) the maintenance of experiences. The conduction of a case study (i.e., the development of a house automation system) is described with focus on reuse and experience-based modification of an effort model. Lessons learned and benefits achieved are discussed. As a basis a model for representing software engineering knowledge is also shown, as well as a prototype implementation of an experience base.*

1 Introduction

Key technologies for supporting quality improvement include modeling, measurement, and reuse of software development knowledge in the form of products, processes, and experience originating from the software life cycle. Software Engineering offers a framework based on an evolutionary quality management paradigm tailored to software development, the *Quality Improvement Paradigm (QIP)* [5]. It is supported by an organizational approach for building software competencies and transferring them to projects, the *Experience Factory (EF) [3].*

The QIP/EF approach has been applied to a project called CoDEx [29] which was aimed at developing a real-time house automation system. The project was conducted in the context of the Sonderforschungsbereich 501 (SFB 501), a long-term strategic research activity. Its goal is to develop and evaluate a set of techniques, methods, and tools that support fast and reliable customization of complex domain-specific software systems. SFB 501 projects are regarded as *experiments* because their main focus is to learn about the development techniques and methods, and there is minor focus on the product development itself. Two types of experiments are considered: case studies (such as CoDEx) and controlled experiments. Case studies (as defined, for instance, in [21], sometimes also called “single project studies” [7]), concern the investigation of a technology (i.e., a technique, method, tool) in a real project. In the context of the SFB 501, the following goals are pursued with case studies: baselines for relevant quality aspects (such as effort, rework, defect slippage) are to be built, causes for process deviations are to be examined, and effects of change are to be measured. In controlled experiments (also called “formal experiments” [21] or “blocked subject-project studies” [7]), we isolate and control the variables of interest. The key discriminator between case studies and controlled experiments is control over the independent variables.

The paper is organized as follows: The underlying model for representing software engineering experiences is surveyed in Section 2. This survey is needed to understand the logical structure of our experience base and its implementation presented in Section 3. In Section 4, the usage and maintenance of experiences and the involvement of the prototype experience base are described in the context of the CoDEx project. The focus is placed on reuse and experience-based modification of an effort model, which is represented in a context-oriented fashion. Some CoDEx project data are added to illustrate the systematic usage and maintenance of this effort model. Section 5 discusses lessons learned and benefits achieved. In Section 6, we compare our work to related research. Section 7, finally, gives an outlook on future activities.

2 Experience Model

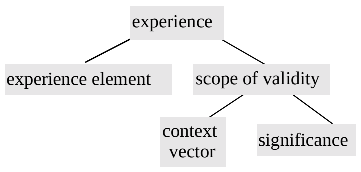


Fig. 1: Experience model

During the execution of SFB 501 experiments, knowledge kept in the prototype *Experience Base of the SFB 510 (SFB-EB)* is used and maintained. For the SFB-EB and CoDEx we created an

1 This work is supported by the Deutsche Forschungsgemeinschaft as part of the Sonderforschungsbereich 501 “Development of Large Systems with Generic Methods”.

experience model (see Fig. 1) which is an adaptation of basic principles of the QIP/EF approach. All kinds of software engineering experience, especially models, instances, and qualitative experience are regarded as experience elements:

Models are a means for explicitly describing real-world experiences and building abstractions therefrom. In the context of the SFB-EB, the following models are mainly used: Development activities (such as design, testing) are represented by process models. Software artifacts (such as documents, code) are represented by product models. Quality aspects (such as effort, defect detection effectiveness) are described by characterizing quality models. Relationships between quality aspects (such as the relation between test effort and design complexity) are described by predicting quality models. Functions, diagrams, pie charts, etc. are used as a means for formally describing quality models. The instrumentation of product and process models for the purpose of applying quality models is represented by GQM models, especially Abstraction Sheets and GQM plans [5, 8].

Instances are software artifacts that are consumed, produced, or modified during the development process. An instance refers to exactly one element in a specific experiment context (in contrast to models that refer to a group of similar elements). Some sample instances stored in the SFB-EB are: products, measurement data, technology packages, and process traces. Products (such as requirement documents) are described informally or in the notation of a description technique in the framework of a specific development approach/method/technique (such as UML, NRL, Statecharts). Measurement data (such as effort data) is captured using forms. Technologies and tools are represented in the SFB-EB via so-called technology packages which support role-specific information necessary for applying the technique/tool. Additionally, technology packages help select the appropriate techniques when setting up a new experiment. Process traces are also regarded as instances. Process traces refer to enactments of the project plan and are documented as sequences of project states or lists of plan deviations.

Qualitative experiences are experiences with models and instances which are documented informally during project enactment or after project completion. Documenting lessons learned is an example of describing qualitative experiences in a structured way as input for subsequent analyses and processing. In the SFB-EB, the following aspects are described: the situation (in which situation did a problem emerge, i.e., what are the project characteristics and the project state?), the symptom (what problem appears?), the diagnosis (what are possible causes for the problem?), the reaction (what are possible solutions for the problem?), the reason (why was a specific solution chosen?), and the result (to which extent could the problem be solved and what were the consequences?). The documentation of the reaction, the reason, and the result is optionally depending on whether the problem solution is important for the success of the running experiment, respectively for subsequent replications or similar experiments.

One of the main ideas of the QIP/EF approach is that an organization should learn from its own business, not from external, ideal models. This implies that the „domains“ in which a particular experience element may be reusable have to be determined. By assessing the extent to which an experiment is a member of the domain, it is then possible to determine whether a given experience element is suitable for reuse in that experiment. To follow this idea for each experience element in the SFB-EB, the scope of its validity is described. The scope consists of a context vector and the significance.

The context vector characterizes the environment in which the experience element is valid. It represents characteristics that may determine whether or not an experience element can be reused or shared within or across experiments. A context vector consists of attribute-value-pairs, e.g., ⟨(project effort, 50.000 h), (application domain, house automation), (programming language, JAVA), (programmer experience, high), …⟩. Although the environment is currently characterized in this simple way, in the future the intention is to use the model-oriented classification scheme for software engineering experiences [6], which combines the flexibility of the faceted classification scheme of Pietro-Díaz and Freeman [25] with the goal orientation of the QIP/EF approach.

The significance describes how the experience element has been validated and to which extent. Our preliminary model for describing the significance of an experience element describes the number of experiments in which the element was involved as well as experimental results concerning the element and their statistical significance (in the case of controlled experiments).

3 A Prototype Instantiation of an Experience Base

Reuse activities in the SFB 501 are based on the EF approach. This section sketches a central component of our instantiation of this approach: the prototype of *the Experience Base of the SFB 501*. First, we describe the logical structure of our Experience Base instantiation in Section 3.1. Then we sketch the basic architecture of our prototype EB implementation (Section 3.2).

3.1 Logical Structure of the SFB-EB

The SFB-EB acts as a repository for all kinds of experience. To provide a basic structure for organizing the Experience Base and allow an iterative setup-process, the Experience Base is subdivided into different modules, called *logical areas*. Each of these logical areas is designed for a special type of experience elements, e.g., technology descriptions or domain-specific knowledge. Within an area, a minimal representation standard for its experience elements is defined. All areas are disjunct and therefore can be integrated into the SFB-EB one after the other.

Just like modules have predefined interfaces to interact with each other, areas have predefined relations. Relations describe how experience elements from different areas should be connected. Some typical relations are ‘experience element X *uses* experience element Y´ or ‘experience element X *is_a* instance of experience element Z´ where X,Y and Z are experience elements of different logical areas. These relations help support the search for experience elements in a given context. For instance, from an experience element that describes a concrete process model one can follow the *uses* relations to the experience elements that describe the technologies addressed within the process model. From the same process model, the *is_a* relation would help find the experience element describing the general process model from which the actual one was derived.

The logical areas and the defined relations between them together form a framework that represents the basic structure of our Experience Base. To provide a better overall structure of the whole framework, logical areas are further clustered into *sections*, with each section containing at least one area and each area belonging to exactly one section.

The benefits of the logical structure are mainly twofold:

1. **Reuse of experience elements.** It helps us reuse the experience elements in the already implemented areas of the SFB-EB. First, because logical areas give an orientation on where to find special types of experience elements, and second, because the defined relations give hints on what else could be reused together with the found experience element, or what else might be used instead.
2. **Collection of experience elements.** It allows us to implement the SFB-EB successively, simply by adding new areas and connecting these experience elements -with the help of the relations- to the already stored experience elements in the other areas of the SFB-EB. In addition, experience elements can be classified with the help of the logical structure *before* they may be integrated into the areas that are already implemented.

Fig. 2 shows the sections and areas of the SFB-EB that have already been defined. Currently we distinguish between two sections named *organization-wide section* and *experiment-specific section.* These sections are similar to the project databases and organization-wide database described by Basili et. al. [1]. In contrast to Basili, we include the experiment-specific section into our EF for reasons of document versioning and traceability of experiments in our research environment. Therefore, we also include the experiment-specific section into our EB. Furthermore, this supports maintaining the organization-wide section since most of these models depend on experience gained within the SFB 501 experiments which are stored in the experiment-specific section.

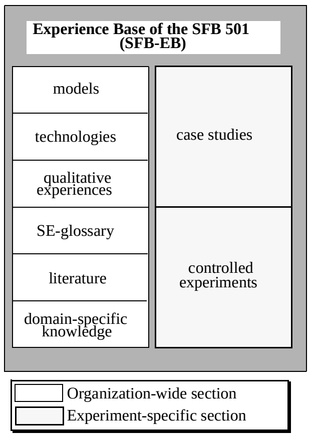

Experience Base of the SFB 501 (SFB-EB)
models
technologies
qualitative experiences
SE-glossary
literature
domain-specific knowledge
case studies
controlled experiments
Organization-wide section
Experiment-specific section

Fig. 2: Sections and areas of the SFB-EB

The organization-wide section stores experience relevant to different variants of experiments (such as generic process models). As of today, it consists of six different areas: the *model* area, the *technologies* area, the *qualitative experience* area, the *SE-glossary* area, the *literature* area, and the *domain-specific knowledge* area. In the *model* area, process models, product models, resource models, and quality models are stored using the process modeling language MVP-L [12].

Some of these models have been adopted from outside the SFB 501 to provide an initial set of experience elements. One example is a MVP-L model describing how to develop real-time systems according to the method of Bræk & Haugen [11]. For different techniques, methods, and tools, technology packages are stored in the *technologies* area. Lessons learned from all experiments are stored in the *qualitative experiences* area. The *SE-glossary*, providing consistent definitions of Software Engineering terminology throughout all experiments, and relevant *literature* also form separate areas in the organization-wide section. Finally, the *domain-specific knowledge* area provides expert knowledge about real-time house automation such as a collection of thermodynamic formulas for heating systems.

The experiment-specific section contains two areas storing the data of *case studies* and *controlled experiments* that have been conducted within the SFB 501. For each experiment, the corresponding experience elements (such as the project plan, the hypothesis, and measurement data) are stored in a separate location. Up to now, the experiment-specific section holds data from seven case studies and two controlled experiments.

Of course, the logical structure of the current SFB-EB prototype is not complete yet. Additional logical areas will have to be added in further iterations. For example, areas for reuse repositories for design patterns and code fragments are currently being defined for the organization-wide section before they will be implemented. The next section describes how the already defined areas are implemented.

3.2 Technical Realization of the SFB-EB

For the implementation of the prototype SFB-EB, the decision was made to use the existing intranet of the SFB 501. Fig. 3 shows the architecture of the prototype implementation of the SFB-EB. It is subdivided into a storage system and a search and retrieval system with the user interface.

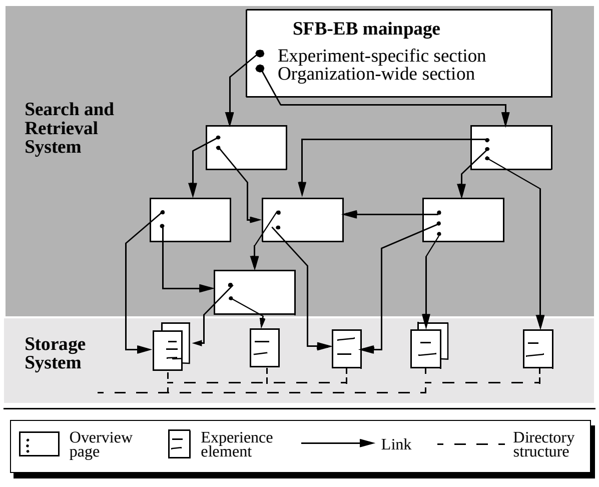


Fig. 3: The architecture of the SFB-EB

The storage system. The logical structure of sections and areas as defined in Section 3.1 is mapped to a directory structure. Two directories in the SFB-EB home directory represent each of the two sections of our logical structure: the experiment-specific section and the organization-wide section. These directories are further refined by subdirectories representing the single areas. The experience elements are stored in these subdirectories.

The search and retrieval system. In the directories of the storage system, HTML pages, denoted as *overview pages*, act as table of contents. An overview page describes the experience elements stored in the (sub)directory and offers links for viewing and/or downloading them. Overview pages can be further refined by links to additional overview pages dealing with special topics. For instance, the overview page for the technologies area is refined by three overview pages: one for tools, one for techniques, and one for methods. With this simple, but effective structure, the basic search and retrieval system of the SFB-EB is realized. In addition to the links of the basic search and retrieval system, overview pages also contain links that represent the relations between the experience elements of the different areas as defined in Section 3.1. For some experience elements, like process models in MVP-L, special plug-ins have been written to start the tools (e.g., [9]) that are needed to view and manipulate them. Since all experience elements are accessible via the overview pages, these HTML pages also form the user interface to the SFB-EB. A more detailed description of our prototype implementation can be found in [16].

4 Experiment-driven Usage and Maintenance of Experiences for Organizational Improvement

This section demonstrates how organizational project improvement is achieved within the SFB-EB according to the Quality Improvement Paradigm (QIP). After a brief introduction to the principles of this paradigm, each step of the QIP is introduced by a short definition instantiated for the context of experiments within the SFB 501. A small excerpt of the case study named CoDEx [29], which was conducted in the SFB 501, serves as an example, giving an accompanying illustration for each QIP step. The excerpt shows the goal-oriented improvement of the quality model '*effort distribution*' and demonstrates how the SFB-EB is integrated into the improvement cycle.

4.1 The QIP

The QIP cycle consists of six steps: three steps that make up the *planning* phase; the *execution* step of the experiment; the *analysis* step, and the last step in which the *packaging* of the newly-gained experience into the organization-wide section takes place (see Fig. 4). The planning phase is further refined into a *characterization* step, a step for defining quality *goals*, and one for instantiating and creating the project plan (originally called the *process* to be executed in step 4). Due to the sequential description of the six steps, the QIP may be interpreted as being executed according to the (idealistic) waterfall execution model. This, however, does not apply to the QIP (as well as to no project at all), e.g., a replanning step may be necessary during the execution phase, which results in one or two steps back while the execution continues. Many scenarios are imaginable that show some form of deviation from the waterfall model.

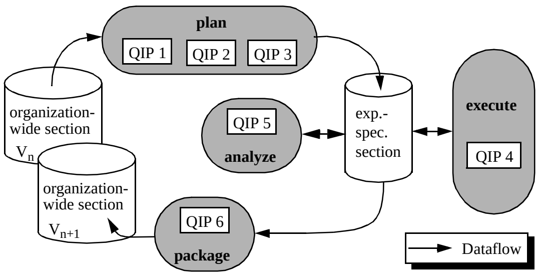


Fig. 4: Involvement of both EB sections in the QIP cycle

The QIP is meant to be applicable to many contexts, e.g., for code reuse or for organizational, goal-oriented improvement of experiences as shown in the following example. The fact that the QIP is described as a cycle indicates that improvement is achieved via a sequence of instances. This explains why the QIP supports organizational improvement though it is applied to one experiment (one of a sequence).

4.2 The CoDEx Experiment

In the SFB 501 context, several experiments were conducted according to the QIP. The example chosen for demonstration is the quality aspect 'effort distribution' of the case study CoDEx. The quality of the effort distribution data, i.e., preciseness/tightness of the hypothesis versa the real data, is one of the major influences on the quality of project management for the planning and execution phase, since it supports the estimation (in the planning phase) and the check (in the execution phase) of time schedule and the total cost.

QIP 1: Characterize

The first step of the QIP defines two activities:
(1.1) Characterize environmental factors for the experiment
(1.2) Select the most suitable artifacts for reuse

In the SFB-EB, every experience element is bound to a *context vector* (see Section 2) consisting of a set of context factors that may differ for the different types of experience elements. By that, each experience element can be characterized and uniquely identified within the SFB-EB. The union set of the context vectors of all experience elements is the context vector needed for characterizing an experiment. If a new experiment is to be planned, it is characterized by the project planner by instantiating the context vector as completely as possible (see center column of Tab. 1). This instance serves as a search schema for retrieving suitable experience elements in the SFB-EB that may be reused. To reduce the number of search hits, the context vector of planned experiments may contain hypothetical entries. One outstanding element to search for in the SFB-EB is a complete experiment. With it, all corresponding experience elements are retrieved, which is useful as long as no specific quality goal is defined.

For CoDEx, the context vector was instantiated completely and the most similar project was chosen as the reuse candidate. Both vectors are shown partially in Tab. 1 with differences marked in grey.

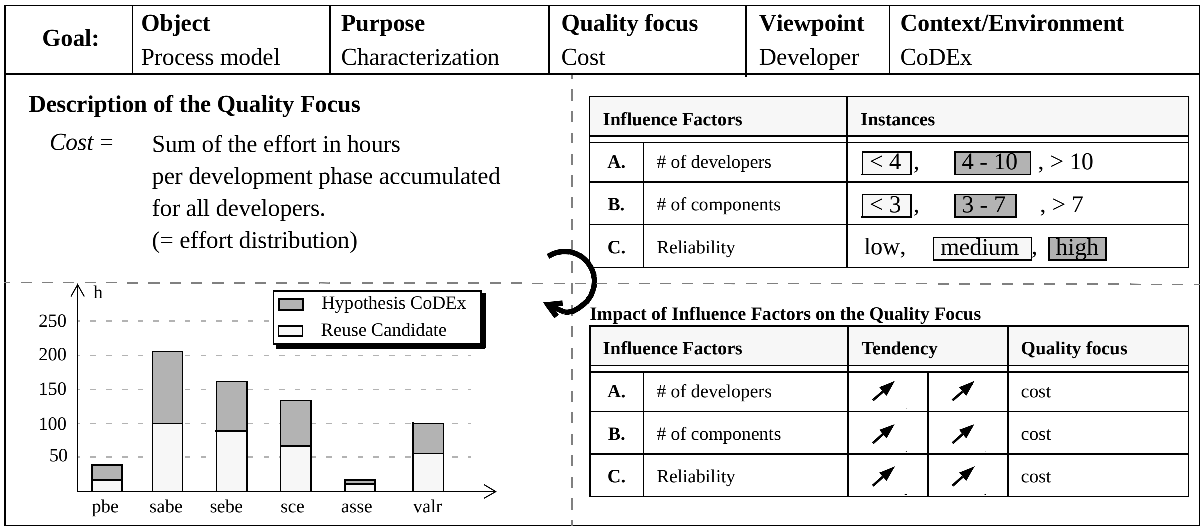
	Context factors	**CoDEx**	**Reuse candidate**
Context vector	Application domain	Reactive systems	Reactive systems
	# of developers	4 research assistants, 1 student	3 students
	Experience of developers	Medium	Medium
	Method of analysis	OMT (with StP)	OMT (with StP)
	# of components	• Building automation system • GUI, …	• Building automation system
	Life cycle model	Waterfall	Waterfall
	Process model	V-model	V-model
	Reliability	High	Medium/low

Tab. 1: Context definition for CoDEx and a search hit (QIP 1)

QIP 2: Set Goals

The second step of the QIP distinguishes three activities:
(2.1) Define the quality goal, and select corresponding models based on the search results of QIP 1
(2.2) Isolate influence factors and their impact on the hypothesis of the quality goal
(2.3) Define hypothesis of the quality goal.

To set up QIP 2 an Abstraction Sheet [20] was used (see Fig. 5). The effort distribution chosen as the sample goal was defined formally according to the GQM paradigm [4] reflected in the top line of Fig. 5. The refinement of the quality focus defined in the top left quadrant is used to fix the type of quality model to be reused. The goal definition also includes the definition of the *object* observed, which, in our case, is the underlying process model on a defined abstraction level. Due to the conformity in the context factor "process model", the reuse candidate (▭ in Fig. 5) exhibits the same process model on the required abstraction level. The corresponding effort distribution was selected as a baseline for hypothesis. Differences in both context vectors (called *influence factors*) pointed out that



Goal:	**Object**	**Purpose**	**Quality focus**	**Viewpoint**	**Context/Environment**
	Process model	Characterization	Cost	Developer	CoDEx

Description of the Quality Focus

Cost = Sum of the effort in hours per development phase accumulated for all developers. (= effort distribution)

Influence Factors		**Instances**
A.	# of developers	< 4, 4 - 10, > 10
B.	# of components	< 3, 3 - 7, > 7
C.	Reliability	low, medium, high

Impact of Influence Factors on the Quality Focus

Influence Factors		**Tendency**		**Quality focus**
A.	# of developers	↗	↗	cost
B.	# of components	↗	↗	cost
C.	Reliability	↗	↗	cost

Fig. 5: Using an Abstraction Sheet for goal definition and hypothesis derivation (QIP 2)

the original effort model had to be adapted (see the different markings in the *Instances* column of the top right quadrant). From the developers' point of view, the functional dependency between influencing factors and the quality focus (see lower right quarter of Fig. 5) shows the tendency that in all three cases of deviation the estimated effort had to be increased. QIP 2 resulted in the hypothetical effort model shown in the lower left quarter of Fig. 5.

QIP 3: Choose Process

The third QIP step defines three activities:
(3.1) Integrate the measurement plan into the process model
(3.2) Instantiate all management models (e.g., process, product, resource, quality models)
(3.3) Create project plan

The coarse-grained process model underlying the effort model was refined and adapted in accordance with the special context of the planned project. In CoDEx this was modeled in MVP-L. The hypothetical effort model was integrated into the process model in a way that an *invariant* was bound to every process (see in Fig. 6). Invariants are of the form

process_id.effort $\le x$, e.g. sabe.effort $\le$ 210 h

and serve as a trigger to show any deviation from the hypothesis. Additionally, every process was instrumented for measurement (see in Fig. 6). This supports the developer in recording his effort on corresponding questionnaires. Every adaptation of the process model and effort model was marked to support traceability of changes.

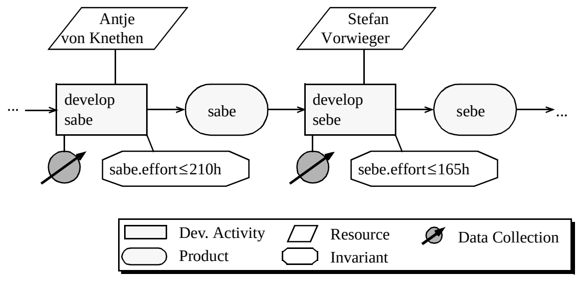


Fig. 6: Graphical MVP-L representation of a partial project plan

QIP 4: Execute

The fourth QIP step includes two activities:
(4.1) Manage the software development
(4.2) Capture measurement data according to the measurement plan

During the development of the software system, effort data were captured according to the project plan via questionnaires, and the process trace was recorded. Since the effort model depends highly on the process model, the process trace is used to additionally explain deviations in the effort model. During the software development in CoDEx,

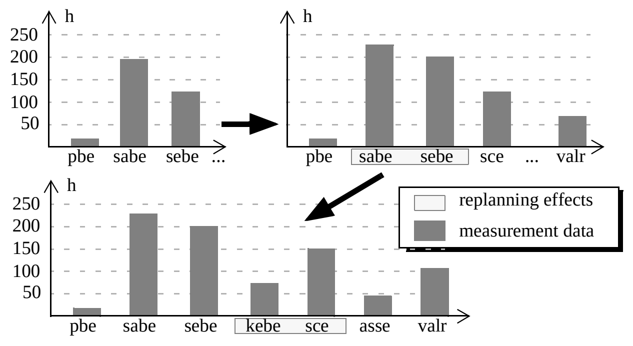


Fig. 7: The process trace: replanning activities, effort distribution

replanning of some activities was inevitable in two major cases shown in Fig. 7. During *sebe*[1], difficulties in the software design were encountered that were interpreted to be caused in a former development activity, a subprocess of *sabe*[2]. The development was set back to this subprocess discarding some results. The second replanning was discovered to be necessary during *sce*[3]. The development was set back to insert a process called *kebe*[4] that had to be completed **before** starting with *sce* and that was not considered in the project plan[5]. Likewise, the effort data for *kebe* had to be gathered for future reuse of the effort model and the process model.

QIP 5: Analyze

The fifth QIP step is refined into three activities:
(5.1) Compare hypothesis with real data and mark deviations
(5.2) Explain deviations
(5.3) Derive proposals for reusable quality models

The analysis of the collected data included the comparison of the hypothetical effort model and measured data as well as the explanation of deviations found between them. A decision had to be made on which deviation could be relevant for future projects. On the basis of this, proposals for new reusable effort models were made. Comparing the hypothesis to the measurement data in CoDEx indicated the following main deviations:

a.) No hypothetical effort baseline for process *kebe* existed (see ¡ in Fig. 8).

b.) The effort was significantly higher for the processes *sabe*, *sebe*, *sce* (see ¿ and ¬ in Fig. 8).

c.) The significant transgression of the effort for *asse* was due to some technical problems, can not be explained by the process trace, and will not be further discussed.

1 German abbr. for 'software system design development'
2 German abbr. for 'software systems requirements development'
3 German abbr. for the 'Coding phase'
4 German abbr. for 'software systems components design development'
5 Note: The CoDEx development has been described as a strict sequential process which only applies to the highly abstracted level of this description. The refined development activities may be dovetailed if entry conditions are fulfilled.

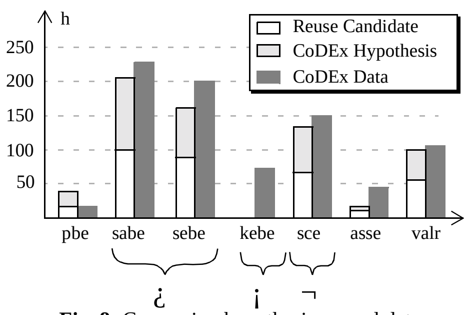


Fig. 8: Comparing hypothesis ↔ real data

The explanation for a.) was given in QIP 4 by the process trace: in CoDEx a new process was supplemented for system development during the second replanning. This was considered a necessary change that should be taken into account for future planning. The deviation of b.) was explained by the two dynamic replanning cycles as explained in QIP 4. They were considered to be unique events. In the first replanning a subprocess of *sabe* (called *sabe-opt*) was omitted. It was not clear whether *sabe-opt* should be executed in future projects or not. Consequently, QIP 5 resulted in two new effort models prepared as entries for the organization-wide section of the SFB-EB. Both models, called *CoDEx eff1* and *CoDEx eff2* (see Fig. 9), show 10% less effort - compared to the measured data - for those processes affected by the replanning, and take the process *kebe* into account. They differ from each other in the way that *CoDEx eff2* considers *sabe-opt* as part of the development process which *CoDEx eff1* does not.

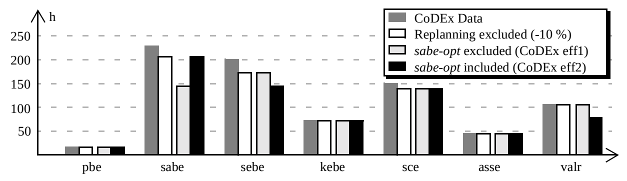


Fig. 9: Deriving two models, ▨ and ■, from the corrected data
▭

QIP 6: Package

The sixth QIP step is made up of four activities:
(6.1) Adapt/supplement context vector, if necessary
(6.2) Adapt/supplement data structure
(6.3) Adapt/supplement relationships between experience elements
(6.4) Synthesize generic models (including significance)

The adapted models *CoDEx eff1* and *CoDEx eff2* needed to be integrated into the SFB-EB to be available for reuse. Due to the discovery of a new influence factor, the context vector had been extended with a new context factor: “Include subprocess *sabe-opt*”(see upper left of Fig. 10). For *CoDEx eff1*, this factor was instantiated with “no”, for *CoDEx eff2* with “yes”. Related effort models in the SFB-EB (such as the reuse candidate) were given the default value “yes” because they were performed with *sabe-opt* (see Fig. 10). The data structure of the organization-wide section had been updated: Fig. 10 shows a generated search tree before and after the inclusion of the newly-gained models. Version V_n shows the leaf level after the search query with the given CoDEx context vector. By integrating the new models, two additional levels were supplemented: a decision point Ⓐ that selects between the ‘old’ and the new models and –on a lower level– a decision point Ⓑ discriminating both new models. Finally, relationships had to be set up between both EB sections (e.g., the “reuse candidate’s” effort distribution *was used* in CoDEx) and the significance entry was updated for the reused and the newly-gained experience elements.

5 Lessons Learned and Benefits Achieved

Due to the basics of the EF/QIP approach, reuse and maintenance of experience could be performed in a systematic and explicit manner. Experiences with CoDEx and other SFB 501 experiments showed that it was difficult to formulate an appropriate context vector because many of the

Influence Factors		Reuse candidate	CoDEx eff1	CoDEx eff2
A.	# of developers	< 4	4 - 10	4 - 10
B.	# of components	< 3	3 - 7	3 - 7
C.	Reliability	low	high	high
D.	Include *sabe-opt*	no	no	yes

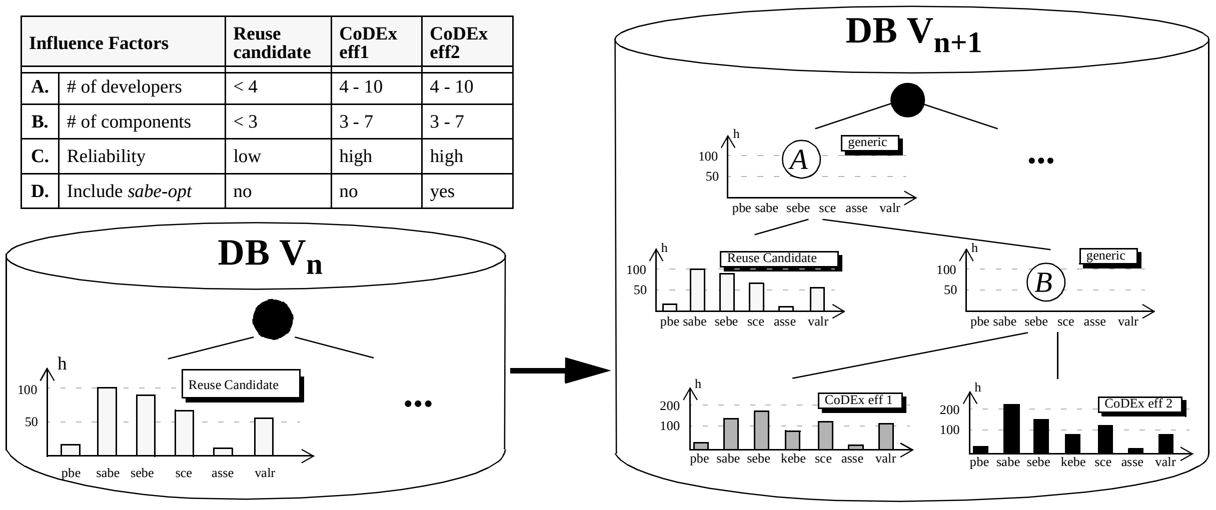


Fig. 10: Visualizing a general retrieval structure of the organization-wide section

factors which may influence the experiment were only assumed to be relevant by subjective judgement. Context vectors of similar experiments in the experience base gave a useful orientation. Planning and goal setting were very well supported by the availability of baseline models in the SFB-EB. Additionally, deviations in the context vectors gave valuable hints on how to tailor these models to the specific experiment characteristics. The exact definition of measurement goals and activities allowed precise acquisition of the relevant measurement data. Non-anticipated events (such as replanning activities) should be carefully documented, e.g., by a process trace, so that they can be considered during analysis. An analysis of the threats to validity must be performed to become aware of the unknown factors that may influence the results without our knowledge. Problems that prevent generalization of the results should also be identified. The packaging revealed a lack of appropriate mechanisms to cope with variants of experience elements. Models should be extended with (empirically-derived) guidelines on how to adapt the models to experiment-specific goals and characteristics. Appropriate representation styles for generic knowledge and suitable tailoring mechanisms are needed.

6 Related Work

The description of related work focuses on frameworks for storing and retrieving software engineering experiences. A comparison with knowledge acquisition approaches (like data mining, descriptive modeling, elicitation) or quality improvement approaches (like Plan-Do-Check-Act Cycle, SEI Capability Maturity Model, Lean Enterprise Management) is beyond the scope of this paper. A comparison of the QIP/EF approach with other quality improvement approaches can be found in [2]. Approaches for storing and retrieving software engineering experience can be roughly classified along the process integration dimension in the following way:

Pure Object Repositories. These repositories offer only simple storage and retrieval mechanisms which are usually tailored to the specific purpose of the repository (like code library, dictionary, measurement database). The internal structure is mainly driven by technical aspects of the system. These repositories can often manage nothing but homogeneous objects such as typical reuse libraries. An example for a web-based, domain-oriented reuse repository is the WSRD library [26] which contains free and commercial products including documents, software components, vendor advertisements, and information on software reuse practices. The WSRD is organized by domains and collections. There are over 1,000 products catalogued by subject area or application in over 35 domains. The products are cross-referenced to more than 10 collections organized by product origin. An example of a data management system that handles corporate, heterogeneous data is the Platinum MVS repository [23]. It provides information such as where data is located, who created and who maintains data, what application processes the data drives, and what relationship the data has with other data.

Tool-oriented Object Repositories. This kind of repositories offers both, user interfaces for organizational and technical roles as well as interfaces to CASE tools. Hence the retrieval and storage mechanisms can be linked to tools supporting the development process. Usually, the collaboration of the repository is limited to just one or to only a few CASE tools. A prerequisite for the tool-based manipulation of the objects in the repository is that the internal storage structure fits the database requirements of the CASE tool. One example is the Microsoft Repository [10]. It is composed of two major components: a set of ActiveX interfaces that a developer can use to define open information models, and a repository engine that is the underlying storage mechanism for these information models. The tool interface is implemented through the Tool Information Model (TIM) which is an object model stored in the repository. It provides an interface for data type definition and data management activities.

Process-integrated Object Repositories. These repositories are integrated in a process-sensitive software engineering environment. This means that the retrieval and storage of objects can be controlled by a process machine which offers the appropriate access/storage interfaces and provides the needed objects to the involved persons at the right time. Ideally, such process-integrated object repositories should be compatible to the process machine and to all the CASE tools used in the development process. In practice there is a lack of joint object management for different tools. Many commercial CASE tools contain their specific internal object management which is not openly accessible to the environment. Hence the benefits of a process machine are basically reduced to coordination and control support. Because of the different roles and activities which are supported by process-integrated object repositories, they must be able to handle heterogeneous objects. A sample instance for a process-centered repository is the Software Management Environment (SME) [15, 18]: a set of data, tools, manuals, and analysis techniques supplied to the project management in order to control the execution of a project, compare it with similar ones, detect and analyze problems, identify solutions [3]. Recent work concentrated on the Web Measurement Environment (WebME), which, in addition, allows for changing data in a distributed, cooperative environment.

The three above-mentioned classes can be seen as major steps towards reuse-based support for software development. Many existing approaches can only gradually be assigned to one of these classes because they do provide the functionality for implementing tool- or process-integration, but without having realized it yet in practice. In the following part, a selection of relevant approaches for storing and retrieving software engineering experiences is shown, which cover one or more of the above-mentioned repository classes.

Within the ***Arcadia project*** [14], several object management systems [17, 28, 30] have been developed to support the various needs of the Arcadia process-centered software engineering environment. Triton [17] is one of those object managers, designed for exploring the needs of process-related activities. Heimbigner describes Triton in [17] as follows: "*It is a serverized repository providing persistent storage for typed objects, plus functions for manipulating those objects*". Therefore it can be classified as a process-centered object repository as well as a tool-oriented object repository. The whole system is based on the Exodus [13] database system toolkit to avoid the cost of new construction from scratch. For the same reason we decided to use Web servers and browsers as a basis for the SFB-EB. Although the main requirements of Triton deal with different aspects (e.g., to support the process coding language APPL/A [27]), there are some requirements that are similar to the ones for the SFB-EB [16]. To allow different tools to access the data stored in Triton, Remote Procedure Calls (RPC) are used for communication between clients and server. The Triton server offers a procedural interface to its clients, acting as a kind of library of stored procedures. With the help of these procedures the dynamic definition of schema elements is reached. Like our logical structure (described in Section 3.1), Triton allows incremental extension of the schema of the object management system. But this can be done by all client programs using the schema definition language. This is against our idea of a central server for maintaining the experience factory and its experience base. In contrast to our instantiation, Triton uses a homogeneous storage schema to provide efficient representation for the wide variety of software artifacts. This means that all data that is shared by programs written in various languages have to be converted to the Triton schema.

The ***Experience Factory*** approach [3] we considered for setting up our experience database is a comprehensive organizational approach for building software competencies and applying them to projects. The following essential characteristics of this approach are realized to a certain extent in the SFB-EB: the reuse of all kinds of experience, a comprehensive incorporation of reuse in the development and maintenance process, the identification and analysis of reuse potential in experience, the definition of the appropriate reuse context and the packaging of experience for easy reusability. Other characteristics have so far been realized only implicitly, especially the establishment of a separate organization to support the reuse and the integrated clustering of experiences to so-called Experience Packages. A characteristic we still have not fulfilled is the formalization of processes for learning, analyzing, packaging, storing, tailoring, and retrieving experiences. In [3] a set of examples of experience factories (e.g., the SEL approach) is described. The focus of many of these examples is more on implementing the organizational framework than on specifying and realizing a structure for an experience base. Hence pure object repositories which are able to store heterogeneous elements are predominating. Various other implementations for accumulating software experience (e.g., the STARS program, the RAPID Center, Japanese software factories) are mentioned.

An Artificial Intelligence approach for capturing and maintaining knowledge within the software development process is the ***knowledge-based software engineering (KBSE) paradigm***, which is concerned with systems that use formally represented knowledge, with associated inference procedures, to support the various subactivities of software development [24]. The central component of KBSE systems is a knowledge base which should ideally satisfy the following requirements (according to [24]): 1) it should have the necessary knowledge (*completeness*), 2) the knowledge should be faithful to the real world (*correctness*), 3) the knowledge should not be self-contradictory (*consistency*), and 4) the system should have efficient algorithms to perform the inferences for the application (*competence*). A main contribution of the KBSE approach lies in the adoption/adaptation of knowledge representation techniques in order to elicit, package, record, structure, and retrieve software engineering knowledge. Examples for knowledge bases (LaSSIE knowledge base, KITSS system, COMET Knowledge Base) can be found in [24], an example for a knowledge representation scheme can be found in [22], and a method for capturing emerging knowledge and synthesizing it into generally applicable forms is presented in [19]. In contrast to KBSE systems, the SFB-EB does not include inference systems yet. The main reason for this is that the needed level of formality is only partly reached in some logical areas of the SFB-EB. In our opinion, inference mechanisms should be applied wherever useful for reuse, but they will be only valuable to particular experience elements. [24] summarizes some deficiencies/limitations of KBSE systems (e.g., high construction and maintenance costs, efficiency problems, difficulties in understanding inferences for humans).

7 Outlook

Future activities concerning the SFB-EB implementation are twofold: First, we have to add more experience elements into the SFB-EB to provide a larger search space, so the returned sets of reuse candidates will offer more alternatives to choose from. This problem can only be solved by conducting more experiments in the context of the SFB 501. Second, we will have to improve the schema of the SFB-EB. A more specific attribute framework based on our logical structure has to be defined to provide an appropriate configuration management and better access to the single experience elements. Correspondingly, an advanced search and retrieval system based on the factors of the context vectors must be provided. An AI approach, like Case Based Reasoning (CBR), that rates the quality of the retrieved experience elements in terms like “element fits to 90%” would be especially helpful. Also, an object-oriented database system (OODB) or an object-relational database system has to be taken into account to support and/or substitute the storage and search and retrieval system of the SFB-EB. Investigating the quantitative impact of known factors of the context vectors as well as identifying new factors should be a long-term activity.

Acknowledgments

We would like to thank our colleagues from the Sonderforschungsbereich 501 “Development of Large Systems with Generic Methods” (SFB 501) at the University of Kaiserslautern for their contributions to the SFB-EB. Especially, we would like to thank Prof. H. Dieter Rombach for helpful discussions during the conception of the SFB-EB. Last but not least, the assistance provided by Ulrike Becker-Kornstaedt and Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) in reviewing the first versions of this paper was much appreciated.

Bibliography

[1] Victor R. Basili. The Experience Factory and its relationship to other improvement paradigms. In Ian Sommerville and Manfred Paul, editors, *Proceedings of the Fourth European Software Engineering Conference*, pages 68–83. Lecture Notes in Computer Science Nr. 717, Springer–Verlag, 1993.

[2] Victor R. Basili. The Experience Factory and its relationship to other quality approaches. In Marvin V. Zelkowitz, editor, *Advances in Computers, vol. 41*, pages 65–82. Academic Press, 1995.

[3] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

[4] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons, 1994.

[5] Victor R. Basili and H. Dieter Rombach. The TAME Project: Towards improvement–oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[6] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. *IEE Software Engineering Journal*, 6(5):303–316, 1991.

[7] Victor R. Basili, Richard W. Selby, and David H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, SE-12(7):733–743, July 1986.

[8] Victor R. Basili and David M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6):728–738, November 1984.

[9] Ulrike Becker, Dirk Hamann, Jürgen Münch, and Martin Verlage. MVP-E: A process modeling environment. *IEEE TCSE Software Process Newsletter*, 10, 1997.

[10] Philip A. Bernstein, Paul Sanders, Brian Harry, David Shutt, and Jason Zander. The Microsoft Repository. In *Proceedings of the 23rd VLDB Conference, Athens, Greece*, 1997.

[11] R. Bræk and O. Haugen. *Engineering Real-time Systems: An Object-oriented language Methology using SDL*. Prentice Hall, London, 1993.

[12] Alfred Bröckers, Christopher M. Lott, H. Dieter Rombach, and Martin Verlage. MVP–L language report version 2. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1995.

[13] Michael Carey, Dave Dewitt, Goetz Graefe, Doug Haight, Joel Richardson, David Schuh, E. Shekita, and S. Vandenberg. The EXODUS Extensible DBMS Project: an Overview. In Stan Zdonik and David Maier, editors, *Readings in Object-Oriented Databases*. Morgan Kaufman, 1990.

[14] The Arcadia Consortium. 1996 ARPA Project Summary, July 1996. http://www.cs.colorado.edu/ arcadia/arpa.html.

[15] W. Decker and J. Vallet. Software management environment (SME) concepts and architecture. Technical Report SEL-89-003, SEL, January 1989.

[16] Raimund L. Feldmann and Stefan Vorwieger. The web-based Interface to the SFB 501 Experience Base. Technical Report 1, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1998.

[17] Dennis Heimbigner. Experiences with an object manager for a process-centered environment. In *Proceedings of the Eighteenth VLDB Conference, Vancouver, British Columbia, Canada*, August 1992.

[18] R. Hendrick, D. Kistler, and J. Vallet. Software management environment (SME) components and algorithms. Technical Report SEL-94-001, SEL, February 1994.

[19] Scott Henninger. Case-Base Knowledge Management Tools for Software Development. *Automated Software Engineering*, (4):319–340, 1997.

[20] Barbara Hoisl. A Process Model for Planning GQM–Based measurement. Technischer Bericht STTI-94-06-E, Software-Technology-Transfer-Initiative Kaiserslautern, University of Kaiserslautern, 67653 Kaiserslautern, Germany, April 1994.

[21] Barbara Ann Kitchenham. Evaluating software engineering methods and tools, part 1: The evaluation context and evaluation methods. *ACM SIGSOFT Software Engineering Notes*, 21(1):11–15, January 1996.

[22] John Mylopoulos, Ales Borgida, and Eric Yu. Representing Software Engineering Knowledge. *Automated Software Engineering*, (4):291–317, 1997.

[23] PLATINUM Technology, Inc.. PLATINUM Repository, December 1997. http://www.platinum.com/products/dataw/repos_ps.htm.

[24] Premkumar Devanbu and Mark A. Jones. The Use of Description Logics in KBSE. *ACM Transactions on Software Engineering and Methodology*, 6(2):141–172, April 1997.

[25] Rubén Prieto-Diaz and Peter Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.

[26] The ASSET staff. Reuse Library, December 1997. http://www.asset.com/WSRD/indices/domains/REUSE_LIBRARY.html.

[27] Stanley M. Sutton, Jr. *APPL/A: A Prototype Language for Software–Process Programming*. PhD thesis, University of Colorado, Boulder, CO, August 1990.

[28] Peri Tarr and Lori A. Clark. PLEIADES: An object management system for software engineering environments. In David Notkin, editor, *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 56–70. ACM Press, December 1993. Published as ACM SIGSOFT Software Engineering Notes 18(5), December 1993.

[29] Stefan Vorwieger. CoDEx: An example case study in the SFB 501 SE-Laboratory (in German). Technical Report 9, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1997.

[30] Jack C. Wileden, Alexander L. Wolf, Charles D. Fisher, and Peri L. Tarr. PGraphite: An Experiment in Persistent Typed Object Management. In *Third Symposium on Software Development Environments (SDE3)*, 1988.