Applying Roles to the SFB 501 Experience Base

Raimund L. Feldmann, Michael Frey, Marco Habetz

SFB 501 Bericht 02/2000

Applying Roles to the SFB 501 Experience Base

Raimund L. Feldmann, Michael Frey, Marco Habetz
{feldmann, frey, habetz} @informatik.uni-kl.de

Sonderforschungbereich 501
Technical Report 02/2000

Software Engineering Group

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

ABSTRACT

For transferring existing knowledge into new projects, reuse has become an important factor in today's software industry. However, to set reuse into practice, reusable artifacts have to be stored somewhere, and must be offered to (re-)users on demand. For this purpose, advanced reuse repository systems like, for instance, instantiations of the Experience Base concept, are quite frequently used. Many people, from different projects, have to access such a repository at various phases of software development processes to retrieve or store reusable data. In order to fulfill the given tasks, each of these users has specific needs. Taking this into account, a reuse repository has to offer tailored user interfaces and functions for different user groups. Furthermore, since the contents of such a repository usually represent the state of the art of an organization's (core) competencies, not everyone should be allowed to freely access each and every repository entry. This is especially true for persons that are not part of the organization.

This report discusses role concepts that can be applied to reuse repository systems to overcome some of the stated access problems. Commonly used roles for software development and reuse repository management are listed. Based on these roles, a basic set of roles, as implemented in the SFB 501 Experience Base, is introduced.

Keywords

roles / reuse / experience management / experience base / SFB-EB / access rights

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

Reuse has become an important factor in today's software industry for transferring existing knowledge into new projects. Consequently, many different reuse approaches have been developed, like patterns and frameworks [10, 11, 20, 25] or product line families [6]. Common to most approaches is the fact that reusable elements have to be systematically collected and stored somewhere in order to be offered to (re-)users on demand. As a result, advanced reuse repository systems like, for instance, instantiations of the Experience Base concept by Basili et. al. [4], are often suggested to face this problem. Some reuse repositories offer large selections of different types of reusable elements [31], while others place their focus only on certain aspects like, for instance, reuse of software processes [14], software components [13], or measurement plans [12]. However, just storing the knowledge in a repository system is not enough. There are many questions that must be addressed in order to design an effective reuse repository, among them:

- *What kind of user interfaces do we need? How should users retrieve or store the repository's data?* – Questions like these address the fact that usually a large number of people, from different projects, have to access such a repository at various phases of software development processes to retrieve or store information. In order to fulfill the given tasks, each of these users has specific needs that are not necessarily satisfied by the complete repository but only by a small part of its contents. Hence, specialized interface, which addresses and offers only the parts and functions of the repository needed for the current task, should be offered. Note that this kind of problem can be compared to the area of process modeling, where multiple views of a given process can be extracted that only focus on the actual part of the process that is relevant for the user [27].
- *How secure is the repository? Can the stored data always be trusted?* – These questions arise because the contents of an advanced reuse repository should always represent the latest state of the art of an organization's (core) competencies. Consequently, not everyone should be allowed to freely access each and every repository entry. This point not only includes a protection against accidental deletions or unwanted usage of experience by people from outside the organization. It also addresses the establishment of a trusted set of reusable experience data where (re-)users can be sure that repository entries are only stored after careful analysis and packaging for reuse. Hence, not everyone should be allowed to add, change, delete, or maintain the entries of such a repository.

Questions like these are not new to the database community. They are quite frequently answered with specialized role-base access concepts [e. g., 19, 24]. But are the given answers applicable to any given reuse repository system? Is there something like a general role concept for reuse repositories? We have been facing these questions at the University of Kaiserslautern. There, the Sonderforschungsbereich 501: 'Development of large Systems with Generic Methods' (SFB 501) a long-term strategic research activity funded by the Deutsche Forschungsgemein-

schaft (DFG), was established in 1995[1] [2]. Since then, several reuse approaches have been developed and tested. To store the gained knowledge, the SE Laboratory project group A1 develops and maintains a reuse repository system, called *SFB 501 Experience Base* [8]. Because the SFB 501 Experience Base (SFB-EB) must be easy to adapt and tailor to the specific needs of all participating research groups, we had to define and introduce a set of basic roles to support those activities. Our set of roles takes into account various aspects of software development processes (e. g., coding, quality assurance, project management) as well as tasks to be performed to run a reuse repository (e. g., adding or maintaining elements of the repository). Those roles act in the following dimensions:

- Allowing flexible role-based process model descriptions for activities to be performed with the repository according to newly developed approaches of single research groups in the SFB 501
- Preparing predefined/specialized user interfaces according to the specific needs of groups of repository users to optimally support them in performing their tasks (e. g., analyzing, adapting, or maintaining knowledge of the repository)
- Simplifying the definition of measurement programs for quality assurance for specific tasks (of certain user groups)
- Granting of access rights according to user groups (e. g., members of a certain research group, certain tools, guests accessing the repository via public web pages) to protect the stored information

Our goal is to make them powerful yet adaptable and abstract enough so that they can be mapped to other role descriptions. Being that flexible, we believe that our set of basic roles can serve as a general role concept applicable to Experience Base systems in general.

The remainder of this paper is organized as follows: Section 2 introduces our definition of a role and reflects related research in this area. Then, we give an overview of commonly used roles in today's software developing processes (Section 3). Furthermore, we identify additional roles that are needed when reuse-based software development is supported by an advanced repository system. In Section 4, we present the basic role concept as implemented in the SFB 501 Experience Base instantiation. The implemented set of roles is mapped to the previously identified roles. Finally, we summarize our results in Section 5.

1 http://www.sfb501.uni-kl.de

2 ROLES

The concept of roles is not new, it has already been used before the IT business existed. Roles have originally been associated with the field of sociology, which emerged in the 19th century. As defined in the Encyclopedia Britannica [30], sociology examines the causes and effects in social relations among persons and in the intercommunication and interactions among persons and groups. Roles were introduced to describe a "*comprehensive pattern of behavior that is socially recognized, providing a means of identifying and placing an individual in society*". An interesting aspect is that the term *role*, according to the encyclopedia, was derived from the theatrical usage, so that the term itself already indicates that a role is different from a person (in case of the origin of the term, the actor playing the role).

It is hard to estimate when the term role was used in our domain for the first time. However, Wieringa and de Jonge [29] indicate that the term was introduced into computer science by the paper 'The role concept in data models' by Bachman and Daya in 1977 [3].

Today, the use of roles in computer science is widespread. Despite the variety of applications, the way roles are used in contemporary computer science literature supports our view of roles for reuse repositories:

> *A role is an abstract concept. Roles assign semantical meaning, can be hierarchically arranged and define a set of associated activities and permissions.*

Related to the topic of permissions, roles are also used to filter information: It is common to have *views* that are associated with roles and that define what is visible for these roles [27].

In computer science, roles can be used to assign semantical meanings. Frakes et. al. [9], for example, discuss roles of *program objects*. Program objects, as used in [9], does not refer to objects in the object-oriented sense, but simply to entities of the source code, like variables, constants, functions, etc.. They define a program object's role as "*the part the object plays in a program*". The idea of assigning roles to program objects is to use these roles to write "*self-documenting code*". This means that a person who reads the source code should be able to understand it without any additional comments. To achieve this, the programmer should choose self-explaining names of the program objects. For example, a variable that holds the result of the addition of two values would have the role of storing the sum. So, in order to add semantical meaning to it, this variable should be named '*sum*' (which is the variable's role in the computation) instead of '*x*'.

Roles are also used in data modeling. The Entity-Relationship Model (ERM) [7] models the real world as a set of objects, which are called entities, and relations between entities. Entity-Relationship Models are typically used to define the data model for relational databases. Therefore, they also have some significance for reuse repository systems, which are often based on such relational databases. Roles in ERMs are used in the context of relations between entities. They give further information on the meaning of the entities in the relationship. For example, in a 'works for' relationship between entity sets 'person' and 'company', the role of a person in that relationship is employee, whereas the role of the company would be the workplace. Roles become more important if the relationship is reflexive, i. e., if the relationship is between entities of the same type. For example, in a 'rents apartment from' relationship between two persons, one would be the landlord, and the other the tenant. In this case, roles are necessary to distinguish the different entities because they are both from the same entity set 'person'. ERMs are usually represented by Entity-Relationship diagrams (ER diagrams), and roles are indicated by labeling the lines that represent the relationship (for more information see [32]). A similar notion is used in object-oriented modeling as described in Rumbaugh et. al. [23], where an object's role describes its function in an association. In this type of approach, role diagrams are introduced to describe how collaborating objects achieve a common goal. In [20], for example, a "*role represents the view some objects in the collaboration hold on another object playing that role [...], it captures the responsibilities of an object with respect to achieving the purpose defined by the role diagram*". However, in the world of object-orientation (OO), roles are used for various purposes. Sometimes, for example, roles are used to overcome deficiencies in modeling dynamically changing behavior of objects. Kappel et. al. [17] use the term *role modeling* to describe object evolution: Objects should be able to be "*instances of different classes at the same time*" and to "*change their class memberships*" during their lifetime in order to model changing behavior. This context-dependent behavior of objects is also called *role behavior* in the sense that "*an object with changing behavior is playing one or several roles*" [17]. This corresponds to the view of roles as a set of associated activities that are performed by a role.

In software engineering (SE), roles are usually (but not necessarily) related to persons, which means a person assumes a certain role. In the context of verification and validation (V&V), for example, [15] suggests defining roles for each task that has to be performed in V&V. The reason for the explicit definition of roles is to make "*the construction and coordination of a master schedule simpler*". In other situations, roles go beyond the realm of processes. Kivisto [18], for example, deals with roles of developers and the use of roles in process models. Kivisto treats both roles for structuring and managing of software projects, which he calls the organizational dimension, and different roles of the actual software developers (process dimension). Kivisto's view focuses on an object-oriented client/server development model.

Rombach et. al.'s view on roles in SE is more general [22]. The latter is a report about software development environments in the Multi-View Process Modeling (MVP) project and supplies some general ideas on roles in SE. In

[22], roles are associated tasks of one or more persons in a software project. The paper distinguishes between roles that modify a project's state, and other roles that only observe. Rombach et. al. also explain that persons and processes (actions that are performed by these persons) are usually associated by using roles: Persons are mapped to roles and roles are mapped to process descriptions. This clearly shows that the goal of the tasks is not part of the role definition: A role is only a "*set of activities*". A role's goal is only implicitly taken into account, i. e., it is described how a role achieves its goal, not what this goal is. Besides the approach to directly map goals to activities that are modeled, [22] extends this traditional view of roles: The paper also includes observing roles by representing their information needs. This corresponds to the view of roles that can be found in [27] and [28]. Verlage defines roles as a set of associated tasks that are assigned to one or more agents. An agent is "*a person or machine responsible for execution of processes*" [28]. A distinction is made between transformational and observing roles: For example, a developer is a transformational role because s/he creates a new document. On the other hand, a project manager is an observing role because s/he does not create or change documents, but checks if the project is still running according to the project plan. Because Verlage uses the term *agents*, his definition is more general than the definition in [22], which only speaks of persons. However, since automation of tasks in software engineering is not always possible, agents are normally represented by persons rather than machines.

Finally, there has been a lot of related work in the field of role-based access control (RBAC), where roles primarily represent permissions. The interested reader will find a comprehensive description of RBAC in [33], and the conference proceedings of the annual ACM Workshop on Role Based Access Control are an extensive source of information as well. RBAC assigns roles to users of a system. Access rights are grouped by role name, and the use of resources and the operations that a user is permitted to perform are based on the user's role [33]. RBAC also utilizes a role hierarchy, but its hierarchy is based on inheritance, meaning that a descendant in a hierarchy inherits the permissions of its parents. As will be seen later in this paper, this is different from our role hierarchy, which embodies granting and withdrawal of permissions, not (automatic) inheritance.

3 ROLES FOR SOFTWARE DEVELOPMENT WITH REUSE

As seen in the previous section, the concept of different roles is not unique to the domain of advanced reuse repository systems. Assigning roles in order to support different views is a common concept in other Software Engineering disciplines like, for instance, in the field of process modeling [27]. Based on existing Software Engineering process descriptions [e. g., 16, 26], a set of commonly used roles in today's software development process can be derived.

3.1 Common Roles for Software Development

When developing (large) software systems, many persons are usually involved, and a lot of different tasks have to be performed. These tasks can be classified as either technical (e. g., design, coding, testing) or managerial (e. g., project planning, project management, quality management). Taking this into account, we now identify roles for these tasks that can commonly be found in many process descriptions, but sometimes are being addressed by different names:

Technical Roles: Technical Roles are responsible for developing the system. We identify seven main technical roles:

1. Requirements Engineer: The Requirements Engineer transforms user or customer needs into a concrete and comprehensive description of the problem to be solved. This formal description materializes in the form of a set of documents addressing customer requirements, developer requirements, and test cases.
2. Design Engineer (high-level design): The architectural Design Engineer uses the proposed problem solution documents to create a software system architecture. S/he is responsible for defining the system design, the component requirements, and the design of component tests.
3. Design Engineer (low-level or detailed design): The Design Engineer uses the component requirements to create a detailed design of the system components defining their data structures and algorithms.
4. Coder: The Coder uses the components detailed design description to create component code.
5. Verifier: The Verifier verifies if the documents describing the software system (from problem description to component code) conform with each other at each level of the system abstraction.
6. System Integration Engineer: The System Integration Engineer uses the software system description and component code to construct an executable software system. S/he creates executable components and software subsystems as well as the final usable system.
7. Validater: The Validater uses the executable software system and the software system description to check if the behavior of the software system conforms to the software system description.

In addition to the tasks listed above, each role should gather measurement data for quality control and/or process improvement based on, for instance, goal-oriented measurement programs as proposed in [5].

Management Roles. Management Roles are responsible for planning and managing project execution. We identify four main management roles:

1. Product Manager: The Product Manager initiates, evaluates, and controls the consistent change of project information during product (i. e., system) development, delivery, and maintenance. Furthermore, s/he is responsible

for product versioning and configuration management.

2. Project Planner: The Project Planner uses project goals and project characteristics to create a project plan. S/he plans activities for software development, product management, project management, and software quality assurance.
3. Project Manager: The Project Manager uses the project plan and disposable resources to allow successful project execution. S/he is responsible for checking if project activities are being correctly executed. S/he must also solve or migrate possible project problems.
4. Quality Assurer: The Quality Assurer checks if the quality of the products produced by, and processes executed by, the software project is in accordance with the software organization is (explicit or implicit) standards. For that, s/he chooses suitable measures and creates a measurement plan to supervise the project.

Besides the two sets of roles listed above, an additional set of roles are needed to run an Experience Base. These roles do not deal with the project (i. e., system development) itself, but with the seeding, updating, and maintaining of the Experience Base. They will be described next.

3.2 Additional Roles for Experience Bases

In contrast to software development, there are not many descriptions of processes to be performed when running and maintaining a reuse repository or an Experience Base. This makes it much harder to identify common reuse repository roles. However, in [1] Althoff et. al. have identified some commonly used roles for Basili's Experience Base concept [4]. We briefly describe the suggested roles below:

1. Experience Manager: The Experience Manager is responsible for maintaining and improving the quality of experience in the Experience Base. S/he assesses the existing measurement of experience quality and sets new measurement goals. Furthermore, the Experience Manager defines the reuse policy, i. e., what kind of experience (gained during project execution) is to be reused.
2. Experience Engineer: The Experience Engineer is responsible for extracting reusable experience gained during project execution. In addition to this, it is his/her responsibility to provide the development organization with reusable experience. S/he also assists in setting goals for projects, project planning, and experience packaging.
3. Project Supporter: The Project Supporter performs several tasks to support project execution. On the one hand, s/he serves as a consultant for the development organization by providing lessons learned and other forms of key corporate knowledge that are stored in the Experience Base. On the other hand, s/he is directly involved in project execution: developing and maintaining measurement plans and supervising the data collection for the project. Furthermore, s/he s responsible for initiating, planning, and controlling changes in the applied processes for process improvements or solving problems that have surfaced.

4. Librarian: The Librarian runs the Experience Base. S/he is responsible for entering data into the repository, and usually reports to the Experience Manager.
5. Experimenter: The Experimenter designs, supervises, and draws conclusions from experiments aimed at assessing (new) technologies or detecting correlations between parameters of processes or products.

Tab. 3.1 summarizes the identified roles to *run* and *use* an Experience Base for software development with reuse. Based on this overview we will introduce our concept for the instantiation of these roles

.

		common role
Roles for Software Development	technical roles	Requirements Engineer
		Design Engineer (high-level design)
		Design Engineer (low-level or detailed design)
		Coder
		Verifier
		System Integration Engineer
		Validater
	management roles	Product Manager
		Project Planner
		Project Manager
		Quality Assurer
Roles for Experience Bases		Experience Manager
		Experience Engineer
		Project Supporter
		Librarian
		Experimenter

Tab. 3.1: Common Roles for Software Development with Reuse

4 THE SFB 501 EXPERIENCE BASE ROLE CONCEPT

This section discusses Experience Base roles developed at the University of Kaiserslautern in the context of the SFB 501. Section 4.1 briefly introduces the SFB 501 Experience Base (SFB-EB) [8, 21]. Section 4.2 discusses the implemented role concept of the SFB-EB. Section 4.3 correlates the SFB-EB role concept with the roles in software development with reuse as introduced in Section 3.

4.1 SFB-EB

The SFB-EB is a web-based reuse repository based on the Experience Base concept [4] that can be accessed with any web browser[2]. It consists of two main sections [8] that are logically, but not physically, disjunct. One section is the so-called *Experiment-Specific Section* (ESS)[3]. The ESS basically serves as a project database. All documents of a project "P" created during planning, execution, and analysis are denoted as *project-related data* and are stored in a part of the ESS that is reserved exclusively for this project. This exclusive part is called *ESS-P* and is created when a project starts. Project-related data in the ESS-P is only readable for members of the same project until it is *released* (i. e., made public to other repository users). Normally, project-related data is released at the end of a project, but can also be released preliminarily, if demanded. The second section is the so-called *Organization-Wide Section* (OWS). The OWS is the experience repository where reusable experience denoted as *experience data* is stored, which has been analyzed and packaged for enhanced reusability from either project-related data from one of the ESS-P entries of the SFB-EB, or from external sources like, for instance, literature or process descriptions. To avoid the cost of new construction from scratch, a commercially available object-relational database management system (ORDMS) was chosen to serve as a basis for implementing the SFB-EB [21].

4.2 Role Concept of the SFB-EB

Access control to the SFB-EB is always user and role dependent. Hence, each user of the SFB-EB has to assume one of the basic roles listed below. Based on this information permission for the repository's data (i. e., read, write, change or delete rights) are granted by offering specialized user interfaces and tailored functions to the user. A user may be able to assume more than one role, for instance, if s/he acts as a quality manager in project '*ESS-P1*' and as a project manager in project '*ESS-P2*'. Major reasons for using roles to access the SFB-EB compare to the ones listed of Section 1. In addition, another one, not yet mentioned before, is related to the system itself: Regular users should not be allowed to change or alter the system or the underlying database schema. This is a task that is exclusively performed by members of the database group of the SFB 501 subproject A3. They take responsibility in operating, maintaining, and extending the ORDBMS.

2 http://donau.informatik.uni-kl.de:18070/cgi-bin/cgiwrap/expfact/webdriver?MIval=login

3 All SFB 501 projects are regarded as experiments because their main focus is to learn about, and improve, new approaches (i. e., the developed techniques, methods, or tools), and their minor focus is on the (software) product development itself.

We will discuss our role concept according to the aspects of handling project-related data, experience data, and additional roles needed in the context of the SFB-EB instantiation. Each role is described by its tasks, access rights, and access methods.

Handling of Project-Related Data. The roles described here are responsible for performing a project “P” and the manipulation of all data that is created by “P” and stored in an area ‘ESS-P’ (i. e., the project database of “P”) of the repository’s ESS. In general, project-related data consists of two parts: one part is the *actual project data* (i. e., all data that is related to product development like documentation or source code). The other is *analysis data*, gained by analyzing the collected measurement data from the project execution. This distinction is important for the administration of access rights because a project manager must not read unreleased analysis data of his/her project, unless s/he holds the permission of the project team[4]. Unreleased measurement has normally not been accumulated, yet, and would enable the project manager to derive information on project team member's individual performance rather than the overall project team performance. If this information were available to the project manager, the project team members might be tempted to make their performance look better by providing false measurement data.

1. Quality Manager: Each project “P” should have a Quality Manager. Within the project “P” s/he is responsible for all measurement related activities and for editing and analyzing the collected measurement data of “P”. S/he is also responsible for releasing analysis data to the project team and the project manager of “P”. To delegate some of these tasks, the Quality Manager can assign or withdraw rights to/from other users with the help of the Quality Manager Assistant role.
2. Quality Manager Assistant: The Quality Manager Assistant supports the Quality Manager in gaining and editing analysis data of a project “P”. If the Quality Assistant has been authorized (by the Quality Manager of project “P”), s/he can also release analysis data to the project team of “P”.
3. Developer: Each project “P” is executed by a number of Developers. They develop the (software) system, and thereby, produce the actual project data of “P”. Furthermore, they provide measurement data to the Quality Manager or Quality Manager Assistant of “P”.
4. Project Manager: Each project “P” is managed by a Project Manager. S/he coordinates the project Developers of “P” and their permissions (i. e., read, write, change, and delete rights) within the project area ESS-P. Within ESS-P s/he can access and manipulate project-related data in any way, but has only access to analysis data already released by the Quality Manager. S/he is responsible for releasing project-related data to other repository users assigned to roles other than the ones listed so far.

4 A project team consists of all persons involved in the software development project. This comprises every person assigned to one of the common roles for software development as identified in Section 3.1.

Handling of Experience Data. The following roles are responsible for experience data management. That is, they add, maintain, change, or delete the data stored in the organization-wide section (OWS). Furthermore, these roles choose reusable experience from project-related data stored in the experiment-specific section (ESS), interrelates them with already existing experience data before it is eventually packaged for advanced reusability and added to the experience data collection in the OWS.

1. EB Manager: The EB Manager administers the OWS of the SFB-EB. His/her task is consolidating experience from already released project-related data of the ESS into the OWS, maintaining experience data in the OWS, and integrating external knowledge into the SFB-EB. Furthermore, the EB Manager initializes a new project “P” by generating an area ESS-P in the ESS of the SFB-EB. Initializing a new project “P”, the EB Manager has to assign the roles of a Quality Manager and Project Manager for “P” to two users. To delegate tasks, the EB Manager can assign or withdraw rights to/from other users with the help of the EB Manager Assistant role.
2. EB Manager Assistant: The EB Manager Assistant supports the EB Manager.

Additional Roles in the Context of the SFB-EB. While all of the previously described roles are directly connected to project-related data and experience data, the following roles are implied by requirements of SFB 501.

1. DB Administrator: The DB Administrator is the only role that can change or alter the system, or the underlying database schema of the SFB-EB. S/he assumes responsibility for low-level maintenance of the underlying ORDBMS. The DB Administrator is the only one who has direct access (e. g., via SQL statements and/or offered programming interfaces) to the underlying database system of the SFB-EB.
2. Guest: The Guest role is usually assigned to persons outside the SFB 501. S/he has only limited access to selected data of the ESS and/or OWS. The rights for this role are assigned and withdrawn by the EB Manager.

Fig. 4.1 summarizes the SFB-EB roles in a hierarchical role tree. In contrast to role hierarchies in traditional role-based access control, the SFB-EB hierarchy is not based on inheritance, but rather displays possible granting and withdrawal of permissions. This means a parent node in the role hierarchy can grant and withdraw rights to/from its descendants, and not that a descendant automatically has the rights of its parent node.

Tab. 4.1 gives an overview of rights as they are assigned per default to the SFB-EB roles. Rights in *{}* are usually associated with the role, but first have to be explicitly granted when the role is assigned to a new user. Based on the listed rights, the different user interfaces and their functions are tailored for each user who accesses the SFB-EB.

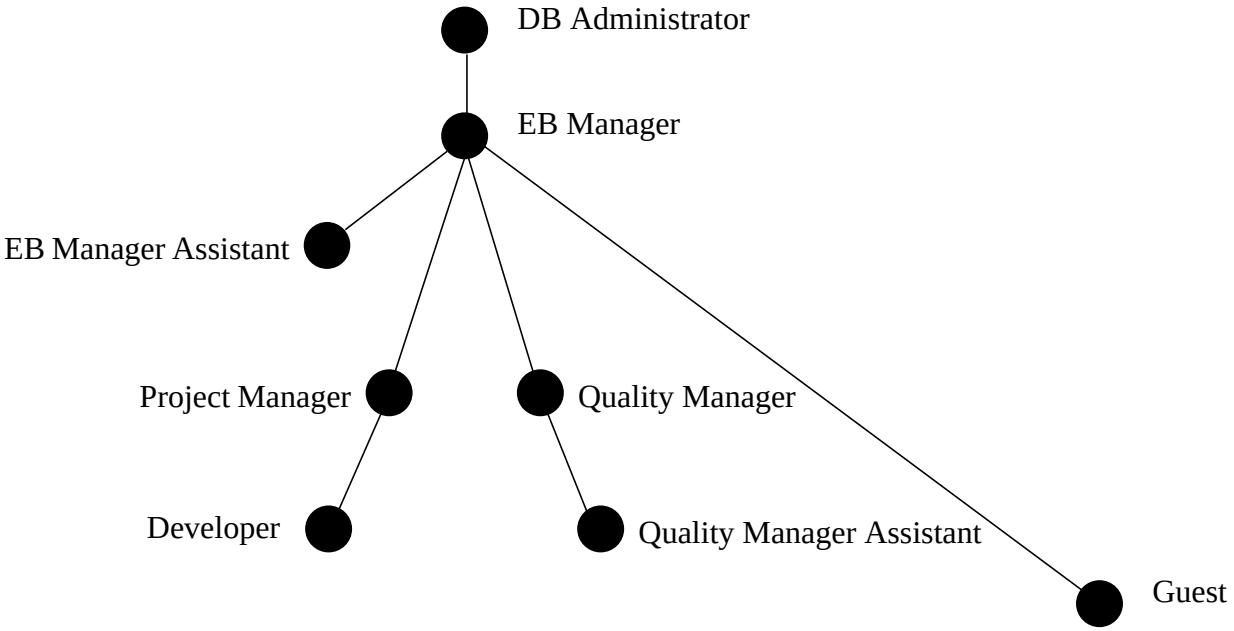

Fig. 4.1: Role Hierarchy in the SFB-EB

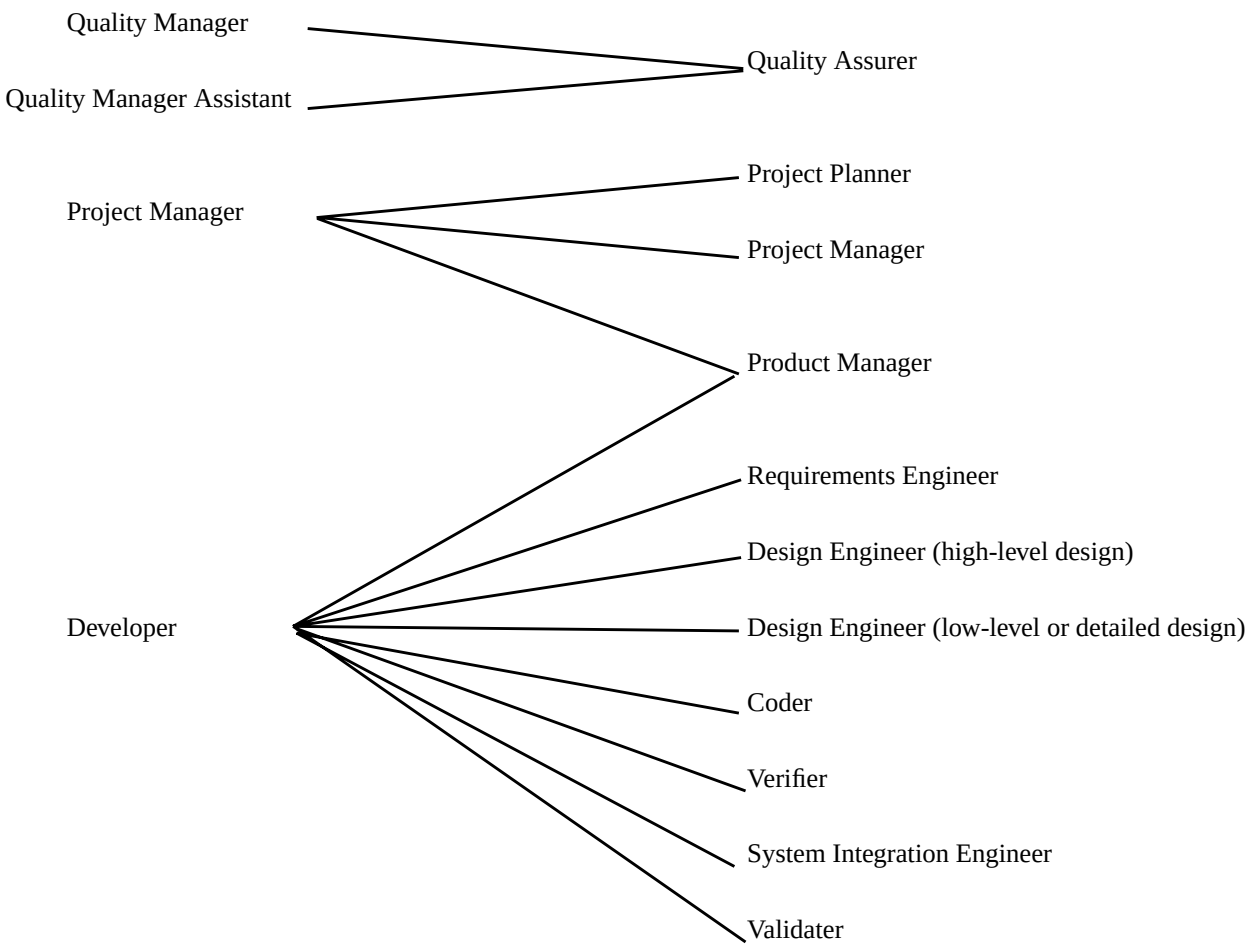

Fig. 4.2: Mapping Common Roles for Software Development and SFB-EB Roles

		SFB-EB roles							
		Quality Manager	Quality Manager Assistant	Project Manager	Developer	EB Manager	EB Manager Assistant	DB Administrator	Guest
OWS		read	read	read	read	read write change delete release grant withdraw	read write change {release}	read	{read}
ESS	ESS-P analysis data (running project)	read write change delete release grant withdraw	read write {change} {delete} {release}	{read}	{read}	{read}	{read}	{read}	
	ESS-P project data (running project)	read	read	read write change {delete} release grant withdraw	read write change delete	{read}	{read}	{read}	
	ESS-P project-related data (finished project)	{read}	{read}	{read}	{read}	read {delete} release	read	read	{read}

Tab. 4.1: Default Rights Assigned to SFB-EB Roles

4.3 Interrelationship Between SFB-EB Roles and Roles for Software Development with Reuse

Common Roles for Software Development. Fig. 4.2 displays the mapping between common roles for software development and SFB-EB roles. The seven technical roles as described in Section 3.1 are all subsumed by the role of Developer in the SFB-EB. Thus, Developers in the context of the SFB-EB can create requirements from the problem description, make designs for architecture, data or algorithms. They can write code, verify or validate documents, and create executable components or systems. From the viewpoint of supporting project execution with reuse in the SFB 501, it is not necessary, and because of the needed flexibility of the role concept, not possible to further refine the software development on this level of abstraction. However, due to the implicit correlation this is

clearly possible, and can easily be implemented for any project “P” based on various combinations of access rights to the project-related data in ESS-P that can be granted to the Developer role by the Project Manager. The activities of Project Planner and Project Manager of the common role model are performed by the Project Manager in the role concept of the SFB-EB. The common Product Manager's activities correspond to activities of both, Developer and Project Manager, in the SFB-EB role concept. In the context of the SFB-EB, planning of product management is usually done by the Project Manager, but the actual execution of versioning and configuration management activities is usually delegated to a Developer. Note that our role concept does not suggest handling the tasks of configuration_management in a distributed manner, but allows delegation and, thereby, distribution of this task, for greater flexibility in tailoring the Project Manager role to specific project needs that may occur. Finally, the common role of the Quality Assurer is mapped to the SFB-EB role Quality Manager and/or Quality Manager Assistant. Again, this is to support greater flexibility in assigning tasks to persons in accordance with specific project needs. The exact task assignment for a given project depends, for instance, on the number of persons involved in the project. As the common role model for software development only deals with tasks to be performed while an project is executed, there is no correlation to the roles of the SFB-EB role concept that are responsible for handling experience data. The same holds for the role of the DB Administrator and the Guest role, because guests are not a common concept for developing software and (database) administration is usually a service provided by a separate group of an organization, central for all projects.

Additional Roles for Experience Bases. Fig. 4.3 displays the mapping of additional roles for Experience Bases and SFB-EB roles. Roles for running and maintaining an Experience Base system (Section 3.2) are mapped to our role concept as follows: Tasks to be performed by Experience Manager and Experience Engineer in the common role concept correspond to activities performed by EB-Manager and EB-Assistant in our concept. The tasks of the Experimenter in the common role concept are subdivided into EB Manager's and Project Manager's tasks. In our context, both the Project Manager and the EB Manager are involved in the development of new experiments to foster improvements of the applied processes, methods, and components. However, this might be different in other organizations. Nevertheless, other solutions that do not suggest distributing these tasks can also be supported by our role concept. The Project Supporter's tasks in the common role concept are threefold. Each task is mapped separately:

a.) The consultant work for the development organization is supported due to the tailored user interfaces for each user accessing the SFB-EB with an associated role. Hence, additional persons are not necessary in our case. However, if additional help is needed, it is provided by specialists of the EB Manager who are assigned to the EB Manager Assistant role.

b.) The development of measurement plans is a task performed by the Quality Manager / Quality Manager Assis-

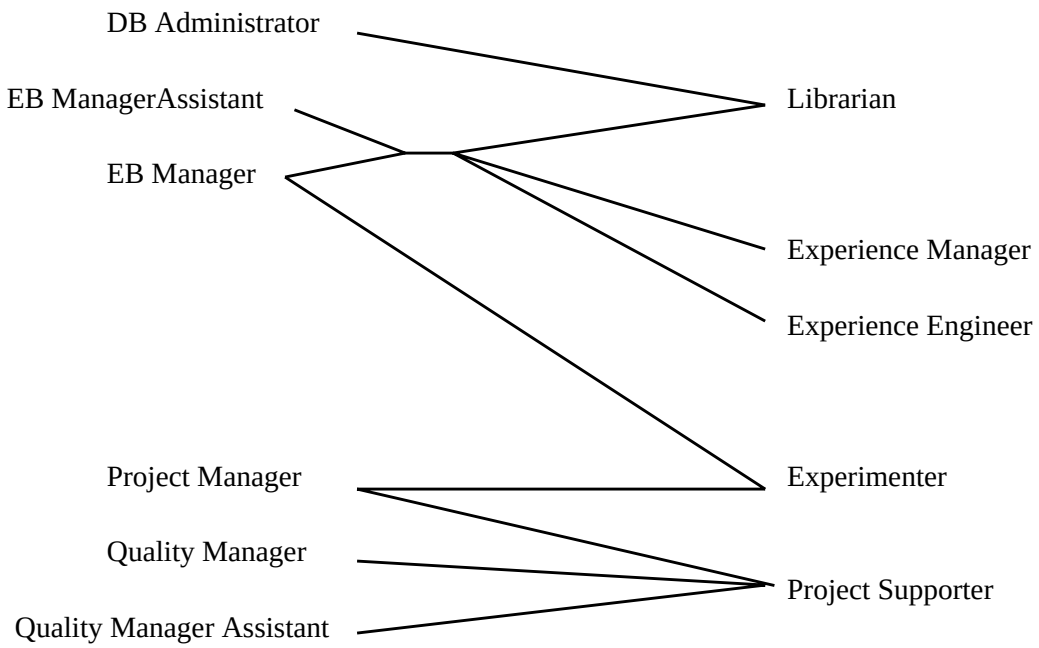

Fig. 4.3: Mapping Additional Roles for Experience Bases and SFB-EB Roles

tant role.

c.) The initialization and planning of process changes within one project is performed by the Project Manager of the corresponding project.

The Librarian in the common roles concept is a combination of EB Manager/EB Manager Assistant and DB Administrator in the SFB-EB role concept: S/he is responsible for running the database and enters experience data into the repository. Although not involved in experience data management, the DB Administrator role is related to Librarian activities because s/he is responsible for running the SFB-EB at a lower level. Furthermore, s/he is the only one who can perform schema evolutions. Tab. 4.2 summarizes our comparison of common roles for software development with reuse and the SFB-EB role concept.

	common roles	SFB-EB roles: Quality Manager	Quality Manager Assistant	Project Manager	Developer	EB Manager	EB Manager Assistant	DB Administrator	Guest
Roles for Software Development — technical roles	Requirements Engineer				X				
	Design Engineer (high-level design)				X				
	Design Engineer (low-level or detailed design)				X				
	Coder				X				
	Verifier				X				
	System Integration Engineer				X				
	Validater				X				
Roles for Software Development — management roles	Product Manager			X	X				
	Project Planner			X					
	Project Manager			X					
	Quality Assurer	X	X						
Roles for Reuse Repositories	Experience Manager					X	X		
	Experience Engineer					X	X		
	Project Supporter	X	X	X			[X]		
	Librarian					X	X	X	
	Experimenter			X		X			

Tab. 4.2: Common Roles of Software Development vs. SFB-EB Roles

5 CONCLUSION

In this paper, we discussed role concepts that can be applied to Experience Base systems. Using process descriptions of today's software developing processes and repository related activities, we identified common roles that address such a system. Based on these roles we introduced a basic set of roles for the SFB-EB instantiation. Access to SFB-EB is always role-dependent. On the one hand, roles are used to customize the user interface of the SFB-EB to the specific needs of a user, and thereby, we optimize access to the stored experience elements of the SFB-EB. On the other hand, roles are used to restrict access to the SFB-EB as well, to guarantee data security. In the context of the SFB-EB, the set of basic roles consists of eight roles grouped according to three categories:

- **Handling of Project-Related Data:**
  Project Manager, Quality Manager, Quality Manager Assistant, and Developer
- **Handling of Experience Data:**
  EB Manager and EB Manager Assistant
- **Additional Roles in the Context of the SFB-EB**: DB Administrator and Guest

The SFB-EB role concept allows for refinement and abstraction, which makes it adaptable to different needs. For example, the Developer role is basically an abstraction of the identified common technical roles for software development as discussed in Section 3.1. Hence, the Developer role can be deliberately refined to the desired degree. The same holds for other roles of the SFB-EB role concept, too. In contrast to most role-based access concepts suggested for databases [e. g., 19, 33], our hierarchical role concept is not based on (automatic) inheritance, but embodies flexible granting and withdrawal of rights.

It shows that the SFB-EB role concept tackles several key problems related to Experience Base systems. By granting data access permission to distinct groups of users, there is much better control of what data is entered into or retrieved from the system. This enables the entered data to be as reliable and secure as possible. Since it takes into account various aspects of software development processes and tasks to be performed on the repository side, it covers a wide bandwidth of activities performed with or on an Experience Base system. This indicates that the SFB-EB roles can serve as a basis for applying and implementing roles in other reuse repository systems, and therefore, might support the reader in selecting suitable sets of roles for their own repository implementation. However, the roles developed for SFB-EB are no panacea. Empirical and experimental evaluation of its use in diverse software engineering environments is still needed. Only then will we be able to have concrete answers on the roles' effectiveness, and external validity of their use in the diverse software engineering reuse repositories.

ACKNOWLEDGMENTS

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). We would like to thank Prof. Dr. Julio Cesar Sampaio do Prado Leite and Manoel Mendonça for their valuable comments on earlier versions of this paper. Dagmar Surmann provided us with deeper insights into the identified common roles for reuse repositories. Our gratitude is extended to our project leader Prof. Dr. H. D. Rombach. Last but not least, we gratefully acknowledge Sonnhild Namingha’s help in improving the paper’s readability.

REFERENCES

[1] K.-D. Althoff, A. Birk, S. Hartkopf, W. Müller, M. Nick, D. Surmann, C. Tautz. Managing Software Engineering Experience for Comprehensive Reuse. *IESE Technical Report 001.99/E*, Kaiserslautern, Germany, Fraunhofer Institute for Experimental Software Engineering (IESE), 1999.

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

[3] C. W. Bachman, M. Daya. The role concept in data models. In *Proc. of the 3rd International Conference on Very Large Databases*, Tokyo, Japan, October 1977.

[4] V. R. Basili, G. Caldiera, H. D. Rombach. Experience Factory. In J. J. Marciniak (ed.), *Encyclopedia of Software Engineering, Volume 1*, John Wiley & Sons, 1994, 469–476.

[5] V. R. Basili, G. Caldiera, H. D. Rombach. Goal Question Metric Paradigm. In J. J. Marciniak (ed.), *Encyclopedia of Software Engineering, Volume 1*, John Wiley & Sons, 1994, 528–532.

[6] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In *Proc. of the 5th Symposium on Software Reusability (SSR)*, Los Angeles, CA, May 1999, ACM Press.

[7] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. In *ACM Transactions on Database Systems, Vol. 1, No. 1*, 1976.

[8] R. L. Feldmann. Developing a Tailored Reuse Repository Structure –Experience and First Results–. In *Proc. of the SEKE´99 Workshop on Learning Software Organizations (LSO´99)*, Kaiserslautern, Germany, June 1999.

[9] W. B. Frakes, C. J. Fox, B. A. Nejmeh. Software Engineering in the UNIX Environment. Prentice Hall Software Series, 1991.

[10] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[11] B. Geppert, F. Rößler. Pattern-based Configuring of a Customized Resource Reservation Protocol with SDL. SFB 501 Report 19/96, Computer Science Department, University of Kaiserslautern, Germany, 1996.

[12] C. Gresse von Wangenheim, A. von Wangenheim, R. M. Barcia. Case-based reuse of software engineeringe measurement plans. In *Proc. of the 10th International Conference on Software Engineering and Knowledge Engineering (SEKE‘98)*, San Francisco Bay, CA, USA, June 1998, Knowledge Systems Institute, Skokie, Illinois, USA.

[13] S. Henninger. Supporting the Construction and Evolution of Component Repositories. In *Proc. of the 18th Intenational Conference on Software Engineering (ICSE’18)*, Berlin, Germany, March 1996, IEEE Computer Society Press.

[14] S. Henninger. An Environment for Reusing Software Processes. In Proc. of the 5th IEEE International Conference on Software Reuse (ICSR5), Victoria, BC, Canada, 1998.

[15] Software Engineering Standards Commitee of the IEEE Computer Society. IEEE Guide for Software Verification and Validation Plans. IEEE Std. 1059–1993, IEEE Computer Society Press.

[16] P. Jalote. An Integrated Approach to Software Engineering. 2nd ed., Springer, 1997.

[17] G. Kappel, W. Retschitzegger, W. Schwinger. A Comparison of Role Mechanisms in Object-Oriented Modeling. In *Proc. of the GI-Workshop Modellierung'98*, K. Pohl, A. Schürr, G. Vossen (eds.), Münster, 1998.

[18] K. Kivisto. Roles of Developers as Part of a Software Process Model. In *Proc. of the 32nd Hawaii International Conference on System Sciences*, 1999.

[19] E. Lupu, M. Sloman. Reconciling Role Based Management and Role Based Access Control. In *Proc. of the 2nd ACM Workshop on Role-Based Access Control (RBAC-97)*, Fairfax, VA, USA, November 1997, ACM Press.

[20] D. Riehle. Composite Design Patterns. In *Proc. of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '97)*, Atlanta, GA, USA, October 1997, ACM Press.

[21] N. Ritter, H.-P. Steiert, W. Mahnke, R. L. Feldmann. An Object-Relational SE-Repository with Generated Services. In *Proc. of the 1999 Information Resources Management Association International Conference (IRMA99)*, Hershey, Pennsylvania, USA, May 1999.

[22] H. D. Rombach, A. Birk, A. Broeckers, C. M. Lott, M. Verlage. Qualitaetsorientierte, prozess-sensitive Softwareentwicklungsumgebungen im MVP-Project (in German). Tecnical Report 256/94, Computer Science Department, University of Kaiserslautern, Germany, 1994.

[23] J. Rumbaugh, M. Blaha, W. Lorensen, F. Eddy, W. Premerlani. Object-Oriented Modeling and Design. Prentice Hall, 1991.

[24] R. Sandhu, Q. Munawer. How to do Discretionary Access Control Using Roles. In *Proc. of the 3rd ACM Workshop on Role Based Access Control (RBAC-98)*, Fairfax, VA, USA, October 1998, ACM Press.

[25] F. Shull, W. L. Melo, V. Basili. An Inductive Method for Discovering Design Patterns from Object-Oriented Software Systems. *Technical Report UMIACS-TR-96-10*, University of Maryland, 1996.

[26] I. Sommerville. Software Engineering. 5th ed., Addison-Wesley, 1995.

[27] M. Verlage. About Views for Modeling Software Processes in a Role-Specific Manner. In *Joint Proc. of the SIGSOFT ´96 Workshops*, San Francisco, CA, USA, October 1996, ACM Press, p. 280-284.

[28] M. Verlage. Ein Ansatz zur Modellierung großer Software-Entwicklungsprozesse durch Integration unabhängig erfaßter rollenspezifischer Sichten (in German). Dissertation, Computer Science Department, University of Kaiserslautern, Germany, 1997.

[29] R. Wieringa, W. de Jonge. The identification of objects and roles - Object identifiers revisited. *Technical Report TR-267*, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1993.

[30] Encyclopedia Britannica. Online. http://www.search.eb.com. January 2000.

[31] The ASSET Reuse Library. http://www.asset.com/WSRD/indices/domains/REUSE_LIBRARY.html. January 2000.

[32] The Entity-Relationship Model. http://www.cs.sfu.ca/CC/354/han/material/notes/354notes-chapter2/node1.html. January 2000.

[33] An Introduction To Role-Based Access Control. http://csrc.ncsl.nist.gov/nistbul/csl95-12.txt. January 2000.