A Methodology for Requirements Analysis and Evaluation of SDEs

Sanjay Dewal

University of Dortmund
Department of Computer Science
Chair for Software Technology
P.O.Box 500500
D-4600 Dortmund 50
Germany

Abstract

Nowadays software systems can be developed for nearly any purpose. For the development of such complex software systems appropriate software development environments (SDEs) are necessary as a "paper and pencil" development is intolerable. For a software producer the introduction of an SDE is part of the technology deployment process during which the staff must learn new methods, gain experience and knowledge by applying the SDE, etc.
This paper focusses on a particular activity of the technology deployment process, namely the selection process (i.e. requirements analysis and evaluation of SDEs). The selection process must (1) produce selection results which are reproducible and comprehensive and (2) be repeatable and flexible in order to be applied for different software producers.
We have developed a selection method which can be applied (1) for a thorough analysis of the requirements of the software producer, (2) for the evaluation of existing SDEs and (3) analyzing the evaluation results. For supporting the method we have developed the environment Requiem.

Keywords: Software Development Environments, CASE, Requirements Analysis, Evaluation, Selection, Reuse, Process Modelling

1 Introduction

Nowadays software systems of varying size and complexity are developed for nearly any purpose. **Software producers**[1] are aware of the fact that it is not sufficient to implement the software system, but also to perform a detailed requirements analysis and a design. As a development using "paper and pencil" only is intolerable, special software systems called **software development environments** (or **SDEs**) have been developed and marketed since many years.
For a software producer buying and introducing an SDE in his environment means that he has to introduce the methods supported by the SDE, train the staff for applying the methods and the SDE, etc. Such a **technology deployment process** takes a lot of time and increase in productivity and/or quality of the software systems produced cannot be expected for a short term. The success of the technology deployment is highly dependent on a "high quality" concept. For instance introducing a highly integrated SDE in an environment where tools such as editor, compiler and debugger are used currently will not be successful. A more convincing concept is to integrate the existing tools in the beginning and to add tools which support early development phases later on.

Thus the technology deployment process is performed repeatedly improving the current technology gradually. Within each cycle an SDE or parts of an SDE are selected (i.e. define requirements, evaluate SDEs, analyze evaluation results, compare software development processes (SDPs)), introduced and applied in the environment of the software producer and the deficiencies during the development are reported (see figure 1).

This paper focusses on a particular part of the technology deployment process, namely the **selection process**. Several high-level requirements on the selection process can be derived from the technology deployment process. The selection results[2] must be **comprehensive** and **software producer specific** which means that the environment and the needs of the particular software producer must be considered. The selection process can be applied **repeatedly** for the evolved requirements. In case a consulting company is performing the selection process it is highly important that the selection results are **reproducible** in order to exclude dependencies to the consulting company.

Existing techniques applied for the selection of SDEs are based on ad-hoc approaches like interviewing experts (see [5, 14, 18, 22, 23]) or evaluation of existing tools (see [12, 21, 2, 15]). The major disadvantage of these techniques are that (1)

[1] i.e. persons or institutions developing software systems

[2] the selection results are the requirements catalogue and the evaluation results of SDEs

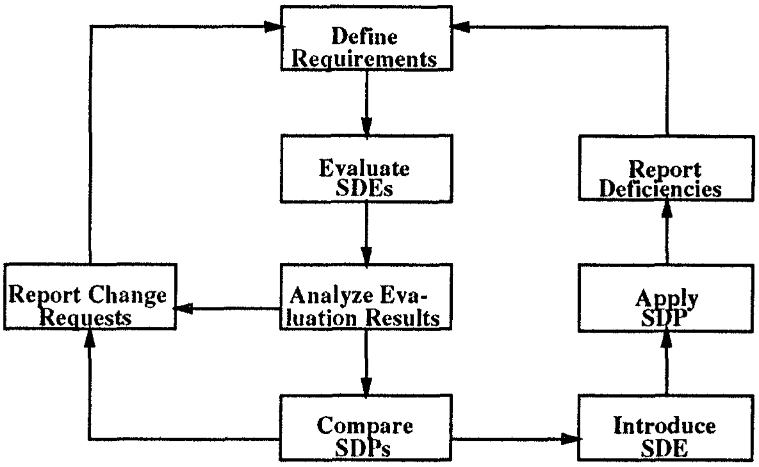


Fig. 1. The Technology Deployment Process

the techniques are applied once and can neither be applied for various software producers (i.e. not reproducible) nor repeatly for evolved requirements, and (3) it is difficult to guarantee comprehensive selection results.

We have developed a method (see [8, 10]) for the selection process which includes (1) the requirements analysis, (2) evaluation of existing SDEs, and (3) the analysis of the evaluation results for purchasing the most appropriate SDE. The definition of (1) a notation for the selection results and (2) a highly-incremental and flexible selection process based on the notation enables a consulting company to repeatedly apply the selection process and to reproduce selection results. Furthermore as various aspects of the environment of the software producer can be considered during the selection process the selection results produced by applying the selection method are comprehensive and software producer specific.

The rest of this paper is structured as follows. In section 2 we briefly describe the selection method. The description includes the key issues of the selection method as well as the particular process which can be applied. In order to provide appropriate support for applying the selection method, we have developed the environment **Requiem**. The architecture and some technical features of **Requiem** are presented in the sections 3 and 4, respectively. In section 5 we present some concluding remarks.

2 The Selection Method

The selection method consists of the high-level activities requirements analysis, evaluation of SDEs and analysis of evaluation results. For each of these activities particular methods have been defined (see [9]) which are briefly presented in the section 2.1, 2.2, 2.3. Some remarks on the complete selection method are given in section 2.4.

2.1 Requirements Analysis

The requirements analysis of SDEs is basically the definition of the software development process (SDP) currently applied and the improvements and extensions of this process planned for the near future. Several process models with various scope for describing such SDPs have been proposed. The process programming language (see [16]) or the graph grammar model (see [9]) can be applied for modelling and controlling the software development process. The formal process model FunSoft nets (see [13]) further allows to analyze and simulate the software development process.

The process models are not very useful for the definition of the SDP with the scope of evaluating SDEs, because elements such as pre- and post-conditions, firing behaviour, time constraints are necessary for the simulation, analysis and control of SDPs only and an evaluation of SDEs regarding such aspects is usually impossible. For the selection process a process model for defining the flow of documents is sufficient. Thus we introduce the **role model**. As process modellers usually connect with the term "activity" the pre- and postconditions as well as subactivities (see [6]), we preferably use the term "role" as a synonym. In the remainder of this section the role model is described briefly.

Role Model. The functionalities and/or tasks of the SDP are mapped onto **roles**[3]. A role is defined by a *rolename* and a *role description* which defines the functionality of the role, the necessary and produced documents, management decisions, non-functional aspects of the functionality, etc. The roles are the main vehicle to define subsets of requirements in the requirements catalogue. All requirements further called **role-based requirements** on the functionality of a role build a subset.[4]. The requirements may define various aspects of the functionality such as the particular functions, necessary and produced documents, management decisions, hardware and software platforms, integration and extension aspects, etc.

[3] Typical examples of roles are the Analyst, Designer, Implementor, Project Manager.

[4] For simplification reasons the definition of a role is extended such that a role consists of a rolename, a role description and the set of role-based requirements

The set of role-based requirements is further structured by defining subsets of role-based requirements called **requirements classes**. Role-based requirements of a requirements class define requirements on particular aspects such as necessary and produced documents, functions, integration aspects and non-functional aspects.

A role can be refined into **subroles** where a subrole covers part of the functionality of another role. The refinement of roles allows the identification of functionalities common in the software development process which are defined as **common subroles**. A typical common subrole is the version manager.

According to the definition of role-based requirements, the refinement of roles implies the refinement of the role-based requirements. Thus it is important to distinguish the importance of different role-based requirements. For instance the requirements on a particular software platform is usually much more important for a software producer than the 50th requirements on the functionality of a text editor. For each role-based requirement a rating which is a value within the interval $[1, 10]$ is defined for indicating the importance of the requested aspect. The rating 10 indicates that the role-based requirements is crucial, while 1 indicates a quite unimportant role-based requirement.

Checking the Requirements Catalogue. The role model allows particular checks within the requirements catalogue. The definition of the notation allows to check the syntactical correctness of the roles. The definition of relations between the requirements classes allows further to check the completeness of a role. For instance the definition of the **uses relation** defined between the role-based requirements on functions and necessary documents allows to check whether any role-based requirement on necessary documents have been defined and whether at least one function needs a necessary document defined. Otherwise such a document becomes obsolete.

For checking the redundancy and consistency of the requirements catalogue it is necessary to standardize and unify the terminology used. We introduce a term dictionary where terms (i.e. functions, documents, etc.) are defined using the method of conceptual structures (see [19]). The term dictionary is used for identifying synonyms, inconsistent usage of terms, etc.

Acquisition of Requirements. With the role model the requirements catalogue can be defined and checked. However, a software producer may not provide the information appropriate for the role definition. It is necessary to define how to acquire the appropriate information from the software producer. The acquisition includes the elicitation of the information by using various types of interviews and observation techniques. The elicited information is interpreted for identifying the terms and for standardizing the terminology of the software producer using conceptual structures.

The interpreted terms are stored in the term dictionary and used as a basis for the definition of roles.

The acquisition process is not only used for the definition of new requirements, but also for validation of acquired requirements.

Reuse of Existing Roles. The role model as well as the standardization of the terminology enables the reuse of existing roles. Reuse of roles is very likely, because many requirements are not software producer specific, but are dependent on the methods applied.

For appropriate reuse the roles are stored in a **global role library** (R-library). The software producer may retrieve roles by defining particular retrieval criteria from a global term dictionary or by navigating through the role hierarchy. As retrieval is based on viewing parts of a role only, it is important that the retrieved roles must be adapted, i.e. role-based requirements may be added, deleted and/or modified.

For enhancing reuse the quality of the global R-library is essential. For improving the quality it is indispensable to update and extend the contents of the global R-library. We ommit the maintenance process in this paper due to space limitations. A detailed definition of the various activities and cases is in [10].

2.2 Evaluation of SDEs

During the evaluation process existing SDEs are evaluated regarding the role-based requirements defined by applying the role model. For a software producer it is usually not sufficient to know whether a particular functionality is provided by an SDE or not, but also to know how "good" the functionality is. More precisely the functional requirements are evaluated regarding different **evaluation aspects**. Each evaluation aspect defines a particular view on the functional requirement defined. Examples of evaluation aspects are *user-friendliness, functional completeness* or *integratedness.* The evaluation aspects are defined in a special requirements class. This approach allows one to define the evaluation aspects for a potential software producer during the requirements definition process.

For the different evaluation aspects it is necessary to define different evaluation techniques. For instance documentation is sufficient to evaluate whether a function exists or not. However, aspects such as user-friendliness can be evaluated after a practical use of an SDE only. We distinguish two different evaluation techniques, namely **analytical evaluation** and **experimental evaluation**.

Analytical Evaluation. The analytical evaluation is based on any documents such as marketing material or user documentation. The aim is to evaluate existing SDEs on the basis of the documents available by using a bivalent measure only. Typical evaluation aspects considered are *functional completeness* and *integratedness.*

Experimental Evaluation. In contrast to the analytical evaluation the experimental evaluation is based on defining and executing a software producer specific experiment. Typically a part of the software systems usually developed by the software producer is developed during the execution of the experiment. There the experiment is defined by precisely identifying the different experiment steps which are executed sequentially. Typical evaluation aspects considered are *user-friendliness* and *performance.* The evaluation results of an SDE regarding role-based requirements using experimental evaluation are presented informally.

Aggregation of Evaluation Results. As the functional requirements are ordered in a hierarchy, it is sufficient to evaluate SDEs regarding **atomic requirements** (i.e. functional requirements which are not further refined). The evaluation results of an SDE regarding non-atomic requirements are aggregated from the evaluation results of the SDE regarding the atomic requirements. As the evaluation results of the analytical evaluation are mapped on a bivalent measure, it is possible to define a straightforward linear additive function for the aggregation. In contrast the aggregation of the evaluation results of the experimental evaluation must be performed manually, as the evaluation results are represented by an informal text.

The evaluation results of the different evaluation aspects are not aggregated, as it cannot be guaranteed that the different evaluation aspects are independent. Thus the evaluation results on an SDE of each evaluation aspect are presented to the software producer.

Exclusion of "unlikely" SDEs. Although many SDEs should be evaluated regarding the requirements of the software producer, it is sufficient to evaluate only "most likely" SDEs to the very detail. The ratings defined for the role-based requirements are used for identifying "unlikely" SDEs. The idea is that crucial requirements must be fulfilled by a selected SDE. This means that any SDE which does not fulfill a crucial requirement is regarded as "unlikely" and is therefore excluded from the further evaluation. Such crucial requirements usually focus on particular functionalities, hardware and software platforms, budget restriction and SDEs can be usually evaluated regarding the crucial requirements very quickly.

2.3 Analysis of Evaluation Results

The evaluation results must be analyzed thoroughly. In particular it is necessary to compare the evaluation results of the different SDEs. Furthermore the SDP defined by the requirements catalogue must be compared with the one supported by an SDE. For this purpose the functionalities, produced and used documents and the sequence of the functionalities are compared. A report is produced for each SDE indicating the particular deficiencies detected during the comparison. On the basis of the report the most appropriate SDE is suggested.

2.4 Remarks on the Selection Method

The role model and the processes defined for the different activities of the selection process allow to reproduce the selection results, to apply the selection method repeatedly, and to produce selection results which are comprehensive and software producer specific. The selection process is very complex and complicated. Figure 2 shows the top-level diagram of the selection process defined using Funsoft-nets (see [13])[5]. The boxes in figure 2 denote the activites and the circles denote the documents. Shaded boxes are refined further by a Funsoft-net.

Thus it is highly recommended to interlink the activities of the selection process and not to perform them sequentially. For instance a result that none of the existing SDE fulfills the requirements of the software producer, can be identified much quicker by interlinking the different activities. After the most appropriate SDE has been identified, the selection results for this SDE are stored in an archive. As the selection results of the chosen SDE describes the environment of the software producer after introducing this SDE, the effort of a repeated selection process is reduced by restoring the selection results and adding the evolved requirements.

3 The Architecture of the Environment Requiem

In the previous section we have presented the major issues of the selection method. Selections made for potential software producers (see [1, 3, 20]) have shown that the method is applicable. However, many data must be maintained throughout the selection process. For instance in [20] 300 requirements have been defined for a tool for the method Structured Analysis (see [7]) only. Five SDEs have been evaluated using five evaluation aspects. Thus around 1500 evaluation results hd to be maintained for each SDE. The huge amount of data and the complexity of the requirements definition and evaluation make it necessary to develop an environment which provides appropriate functionalities for applying the selection method. In this section we describe the architecture of the environment **Requiem**.

All components of the environment are integrated regarding the user-interface. In addition to the integration of the user-interface, data-integration is supported via

[5] Funsoft-nets are high-level Petri-nets.

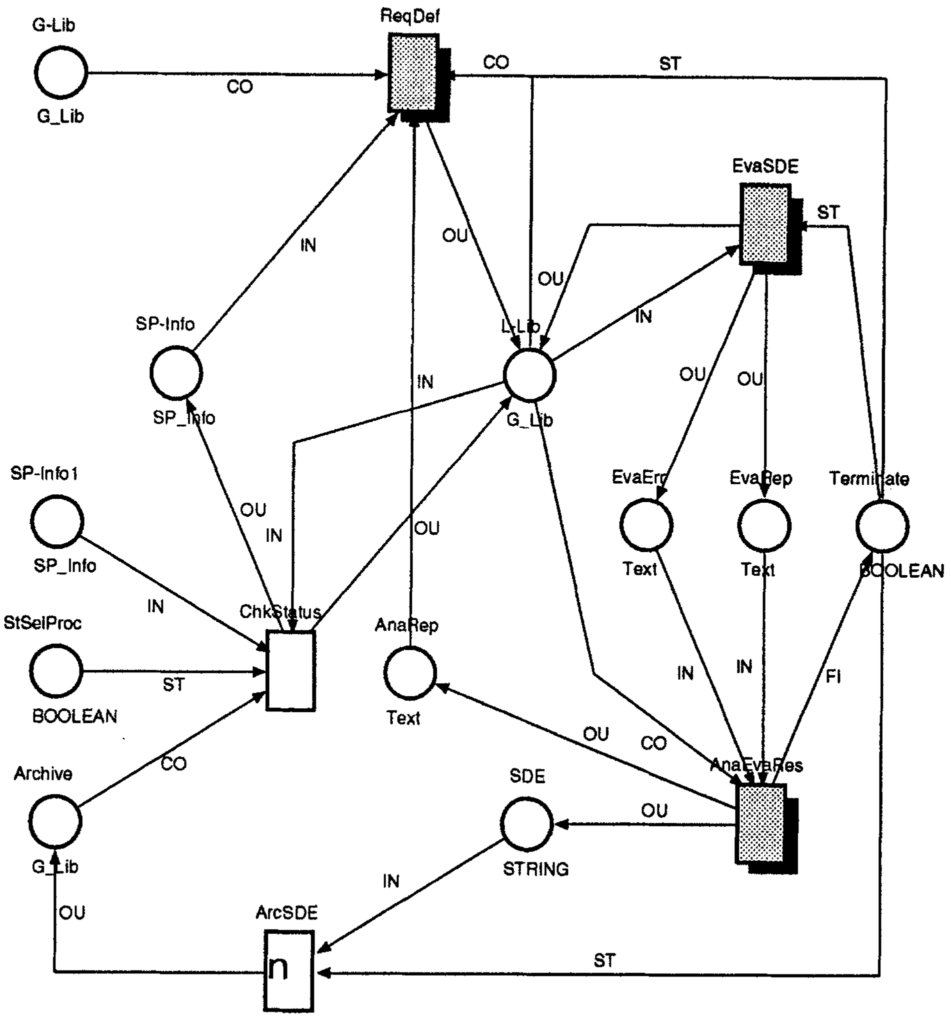


Fig. 2. The Top-Level Diagram of the Selection Process

a common database. All objects produced by the different components are stored in the common database. In figure 3 the complete architecture of the environment is shown. The boxes in figure 3 denote the different components of the environment (in the sense of modules), the arrows denote a use-relationship.

The common services are the user-interface (*X-Windows*) implemented using X-Windows and the database management system (*Oracle*) using a commercially

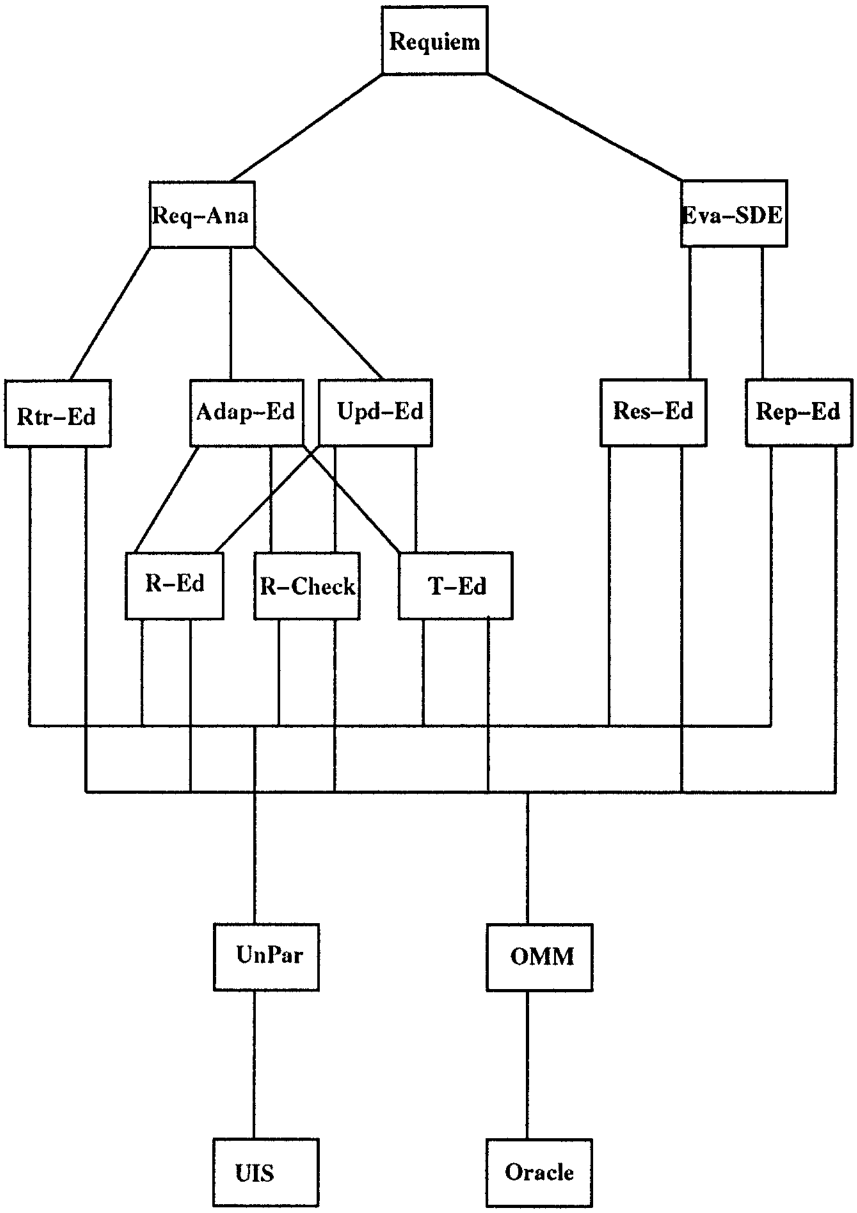


Fig. 3. Architecture of the Environment Requiem

marketed database management system Oracle. The user-interface is accessed by the different components of the environment via the unparser (*UnPar*) and the database management system is accessed via the object management module (*OMM*) which passes all requests of the components regarding objects stored in the database to the database management system.

On top of these components are the components providing the necessary functionalities for the requirements definition process and the evaluation process. A detailed description of the components is presented in [17]. In the following the functionalities provided by the different components are briefly described.

Role Editor (*R-Ed*). The role editor provides the functionalities for maintaining a role. Maintenance of a role means that it must be possible (1) to define a role "from scratch", (2) to modify a role, and (3) to delete a role. In particular it is possible (1) to maintain the role (i.e. the sets of superroles and subroles, as well as the rolename), (2) to view a role, (3) to maintain the role description of the role, (4) to maintain the role-based requirements of the different requirements classes, (5) to maintain the tuples which are elements of the relations defined between the requirements classes of the viewed role.

Term Dictionary Editor (*T-Ed*). The term dictionary editor provides the functionalities for maintaining terms in the term dictionary. In particular the functionalities provided must enable the requirements engineer (1) to maintain the terms, (2) to maintain the concept types, (3) to view terms, (4) to maintain the elements which are elements of the relations defined for terms and concept types.

Role Checker (*R-Check*). The role checker provides the functionalities for checking a single role or for checking a role hierarchy. The checking of the completeness of a single role as defined in section 2 is performed automatically.

In contrast to the checking of a single role, the checking of the role hierarchy cannot be automated. Therefore the functionalities provided by the component which support the requirements engineer must be appropriate and adequate for the checking task. For instance the component provides functionalities to navigate through the role hierarchy and to view each role. Furthermore the requirements engineer must be able to view terms defined in the local term dictionary in order to check the consistency of the terms used.

Retrieval Editor (*Rtr-Ed*). The retrieval editor provides the functionalities for retrieving roles from the global R-library. As described in the section 2 it is necessary to support the point retrieval as well as the guided retrieval. For the point retrieval the component provides functionalities for defining and executing queries. In particular it is possible (1) to view the global term dictionary to find the terms for the query, (2) to create a new query from already defined queries and (3) to view the roles retrieved. For the guided retrieval the component must provide functionalities for (1) navigating through the role hierarchy of the global R-library and (2) for viewing the role descriptions of a role.

Adaptation Editor (*Adap-Ed*). The adaptation editor provides the functionalities for adapting roles in the local R-library and the terms in the local term dictionary. The adaptation of a role means that retrieved roles can be modified or that a role can be defined "“from scratch". Adaptation of a term means that retrieved terms can be modified or that terms can be defined "from scratch". The adaptation editor uses the components (1) *R-Ed* for maintaining a role, (2) *R-Check* for checking a role or the role hierarchy and (3) *T-Ed* for maintaining the terms in the local term dictionary.

Update Editor (*Upd-Ed*). The updating editor provides the functionalities for integrating the roles from the local R-library into the global R-library and for integrating the terms from the local term dictionary into the global term dictionary. There the integration of a role in the global R-library means that the requirements engineer must add the role from the local R-library to an existing role hierarchy into the global R-library. For this purpose the component provides functionalities (1) to navigate through the role hierarchy of both R-libraries (i.e. the global and the local R-library), (2) to view a role in the local R-library and (3) to maintain a role in the global R-library. Similar functionalities must be provided for integrating terms corresponding to integrated roles from the local term dictionary into the global term dictionary. The update editor uses the components (1) *R-Ed* for maintaining a role, (2) *R-Check* for checking a role or the role hierarchy and (3) *T-Ed* for maintaining the terms in the local term dictionary.

Evaluation Results Editor (*Res-Ed*). The evaluation results editor provides functionalities for maintaining the evaluation results for a given SDE. The component furthermore allows the maintenance of different evaluation aspects. In particular the component provides the functionalities for (1) viewing atomic requirements and assessable requirements of a role, (2) to maintain the evaluation results of the SDE regarding the atomic and assessable requirements and (3) to check whether the evaluation results of the SDE regarding all atomic requirements have been defined and whether all evaluation results of the SDE regarding all functional requirements refined from a non-atomic requirements have been defined.

In case an SDE is evaluated using the analytical evaluation, the component provides functionalities to define values (i.e. "0" for not fulfilled and "1" for fulfilled) as evaluation results. Furthermore the component provides the functionality for the aggregation of evaluation results in order to create the evaluation results for the SDE regarding the non-atomic requirements.

In case an SDE is evaluated using the experimental evaluation the component provides functionalities (1) to maintain experiments and to map each experiment step onto one or more atomic or assessable requirements, (2) to check the completeness of the experiment (i.e. are all atomic and assessable requirements covered by the experiment), (3) to maintain the evaluation results of an SDE regarding atomic and assessable requirements as well as (4) to maintain the evaluation results for non-atomic requirements.

Evaluation Report Editor. (*Rep-Ed*). The evaluation report editor provides the functionalities for maintaining the evaluation report presented to the software producer. Furthermore the component provides functionalities for viewing the totally aggregated evaluation results in order to produce and print the evaluation report.

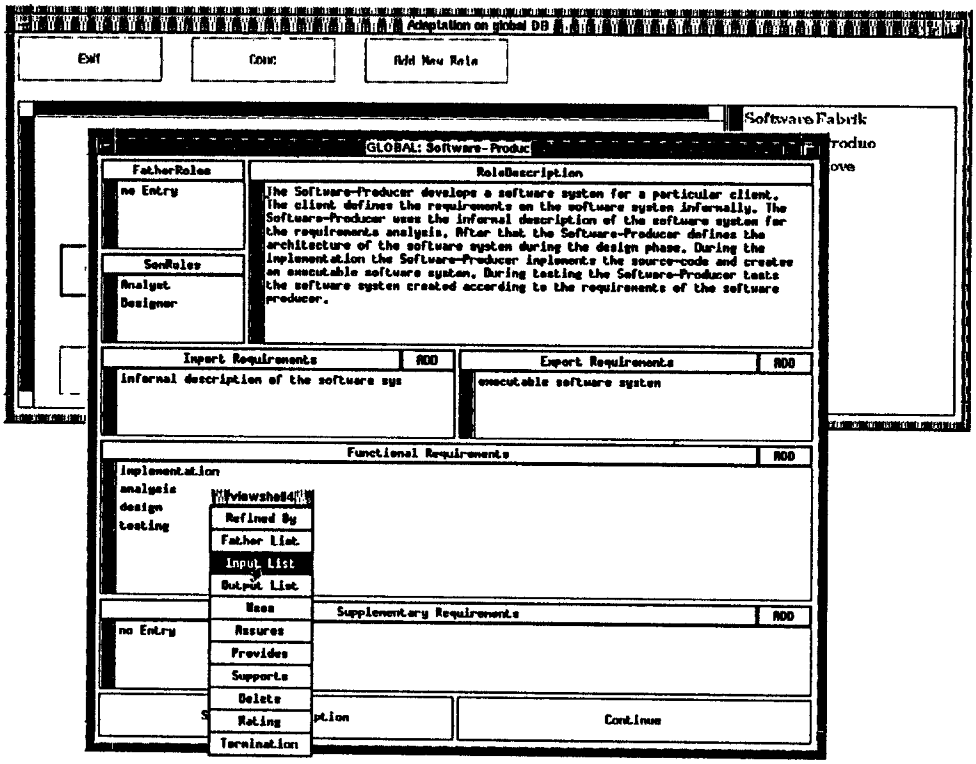
Requiem Control. (*Requiem*). The Requiem control component control the complete environment **Requiem** and provides the functionalities for the invocation of the different components like the adaptation component, or the retrieval component.

Figures 4 and 5 give an idea about how the environment **Requiem** appears to the requirements engineer and the evaluation engineer, respectively. Figure 4 shows a snapshot of a session of the requirements engineer and figure 5 shows a snapshot of a session of the evaluation engineer.

4 Technical Details of the Environment Requiem

The environment **Requiem** has been implemented in C on Sun/Sparc workstations under the SUN OS 4.1.1. The current version of the environment consists of 70000 lines of code. As mentioned before we have used Oracle as the database management system in order to use a commercially distributed database management system which provides functionalities for parallel access as well as for security regarding system crash, etc. For increasing the user-friendliness of **Requiem** we have used the X-windows systems which is state-of-the-art and is widely accepted.

The current version of the environment **Requiem** has been already used to define several requirements catalogues. During the definition of these requirements catalogues we have detected several problems regarding the user-friendliness of the environment. A new version of **Requiem** which is currently developed will thus focus on the improvement of the user-friendliness by installing an improved help system



FatherRoles: no Entry

SonRoles: Analyst, Designer

The Software-Producer develops a software system for a particular client. The client defines the requirements on the software system informally. The Software-Producer uses the informal description of the software system for the requirements analysis. After that the Software-Producer defines the architecture of the software system during the design phase. During the implementation the Software-Producer implements the source-code and creates an executable software system. During testing the Software-Producer tests the software system created according to the requirements of the software producer.

Input Requirements: informal description of the software sys

Export Requirements: executable software system

Functional Requirements: implementation, analysis, design, testing

no Entry

Fig. 4. A Snapshot from a Session with the Requirements Module Editor

and error handling as well as improvements of functionalities for the requirements definition and the evaluation of SDEs.

5 Conclusions and Summary

In this paper we have outlined the selection method and we have presented the environment **Requiem** which provides functionalities for applying the selection method adequately. The selection method has practically been applied for selecting SDEs for various software producers (see [1, 20], etc.). The practical experiments have pointed out that the selection method is appropriate for the selection of SDEs. Especially the possibility of reusing the roles from the global R-library enables the requirements engineer to support and especially guide the software producer during the definition of the requirements. The archiving of the requirements catalogue of a software producer allows the requirements engineer to consider the evolution of requirements of the software producer over a long period of time.

The development of the environment **Requiem** has increased the practicabil-

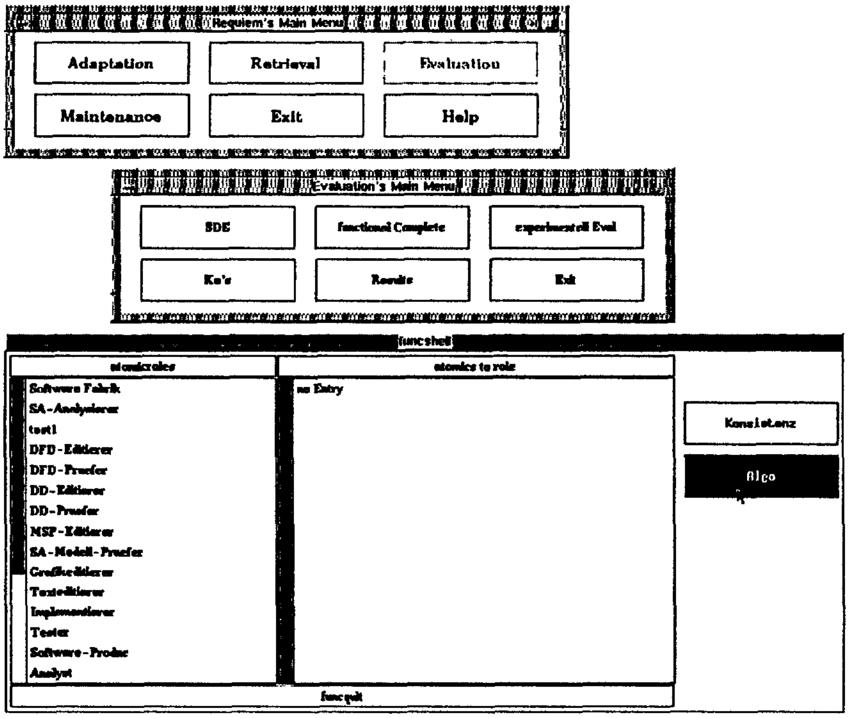


Fig. 5. Evaluation Editor

ity of the selection method, as the "paper and pencil" technique for applying the selection method prevented the use of all advanced features of the method. The first version of **Requiem** has been developed so far and shown that most of the functionalities necessary for supporting the selection method appropriately are already implemented and tested. The second version which is currently developed will focus on the improvement of **Requiem** regarding the missing functionalities and user-friendliness.

Acknowledgements Thanks to the members of the students group, Dirk Altenhoff, Christian Bunse, Hatice Carman, Mirco Gröger, Manfred Hübner, Martin Kampmann, Dirk Ohrndorf, Guido Quelle, Sabine Sachweh, Michael Schmidt and Paolo Secci, for their engagement in the development of the environment Requiem. Thanks also to my guides Prof. Herbert Weber and Udo Kelter for their valuable comments on the thesis.

References

[1] Albrecht, T.: *Evaluation and Comparison of several Tools supporting the Design*, M.S. thesis, University of Dortmund, to appear (in German)

[2] Baram, G., Steinberg, G.: *Selection Criteria for Analysis and Design of CASE Tools*, ACM Software Engineering Notes, Vol. 14, no. 6, Oct'89

[3] Becker, S.: *Evaluation of Software-Engineering-Tools on the Basis of a user-specific Requirements Analysis*, M.S. thesis, University of Dortmund, Jan'89 (in German)

[4] Boehm, B. W.: *Improving Software Productivity*, IEEE Computer, Sep'87

[5] Castor, V. L.: *Criteria for the Evaluation of ROLM*, Corporation's Ada Work Center, Air Force Wright Aeronautical Laboratories, Jan'83

[6] Cheatham, T. E.: *Emerging Issues*, Proceedings of the 4th International Process Workshop, editor C.J. Tully, Moretonhampstead, ACM SigSoft Software Engineer Notes, Vol. 14, Number 4, Jun'89

[7] DeMarco, T.: *Structured Analysis and System Specifica- tion*, New York, Yourdon Press, 1978

[8] Dewal, S., Kelter, U.,Stock, M.: *A Methodology for Requirements Analysis and Evaluation of SDEs*, ESF Seminar, Berlin, Nov'90

[9] Deiters, W., Schäfer, W., Vagts, J.: *Formal Methods for the Description of Software Development Processes*, Internal Memo of the Chair of Software Technology, University of Dortmund, Apr'88

[10] Dewal, S.: *A Methodology for Requirements Analysis and Evaluation of SDEs*, Ph.D. thesis, University of Dortmund, Department of Computer Science, Jan'92

[11] Working Group 8: *ESF Requirements*, Internal Report, Oct'87

[12] Glickman, S., Becker, M.: *A Methodology for Evaluating Software Tools*, Trans. on SE, Jan'85, p.190

[13] Gruhn, V.: *Validation and Verification of Software Process Models*, Ph.D. thesis, University of Dortmund, Jun'91

[14] Houghton, R. C.: *A Taxonomy of Tool Features for the Ada Programming Support Environment (ASPE)*, U.S. Department of Commerce, National Bureau of Standards, Dec'82

[15] Houghton, R. C., Wallace, D. R.: *Characteristics and Functions of Software Engineering Environments: An Overview*, ACM Software Engineering Notes, Vol. 12, no. 1, Jan'87

[16] Osterweil, L.: *Software Processes are Software too*, Proc. of the 9th Int. Conf. on Software Engineering, Monterey, California, Apr'87

[17] Projektgruppe Requiem: *Development of the Environment Requiem*, Final Report, University of Dortmund, to appear (in German)

[18] Schulz, A.: *Ein Klassifizierungs- und Bewertungsschema für Software-Engineering-Werkzeuge, insbesondere CAS- Systeme*, Angewandte Informatik, Vol.28, No.5, Mai'86, p191-197

[19] Sowa, J. F.: *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, 1983

[20] Stock, M.: *Evaluation and Comparison of several Tools for the Method Structured Analysis*, M.S. thesis, University of Dortmund, to appear (in German)

[21] Troy, D. A.: *An Evaluation of CASE Tools*, CompSac'87, 1987, p124ff

[22] Weiderman, N. H., Habermann, A. N., Borger, M. W., Klein, M. H.: *A Methodology for Evaluating Environments*, SigPlan Notices, Jan'87, p199-207

[23] Zucconi, L.: *Selecting a CASE Tool*, ACM Software Engineering Notes, vol 14, no 2, Apr'89

This article was processed using the LaTeX macro package with LMAMULT style