# Databases in Software Engineering: A Roadmap

**Klaus R. Dittrich** **Dimitrios Tombros** **Andreas Geppert**

Department of Information Technology
University of Zurich
Winterthurerstr. 190
CH-8057 Zurich, Switzerland

dittrich@ifi.unizh.ch tombros@ifi.unizh.ch geppert@ifi.unizh.ch

**ABSTRACT**

The development of software systems is a complex process involving a variety of individual tasks, collaborative work, and lifecycle management of the resulting products and product parts. Software development teams are supported in their work by a number of methods for the various production phases, and by appropriate tools of varying degrees of sophistication. Due to the immaterial nature of software, its development can be regarded as the goal-oriented production of a heterogeneous set of mostly highly-structured, interrelated units of information—information that usually needs to be persistently stored and managed to meet various quality and reliability criteria.

Database technology, over the past 30 years, has provided a wealth of concepts, methods, systems and tools that exactly address this problem—albeit for information as it is typically encountered in more business and bookkeeping kinds of environments. Database management systems are in everyday use widely today, and have recently extended their range of application areas to engineering and scientific fields. Thus it looks very promising to investigate how database technology might be beneficially used in software engineering, and indeed many such approaches have been taken during the last decade and a half. To be fair, success so far has been rather limited. In this paper, we summarize the various requirements software engineering poses on database technology, then briefly describe past and present achievements, and discuss the challenges that need to be met in order to better achieve the goals in the future.

## 1 INTRODUCTION

Database technology and software engineering are related in a number of ways. First, software engineering methods and tools have to be used for the construction of software providing database functionality and for the development of database applications. Furthermore, special-purpose software engineering tools and techniques have been developed, for example, in data modeling and implementation of persistent application components. Second, database technology may support through appropriate services the activities, tools, and techniques involved in software development processes. This relationship is the focus of the present paper.

Research in database support for software engineering has been on and off an active research area since the mid-80s. On the one hand, it has been motivated by various advances in database technology, the most prominent one being probably the emergence of *object-oriented database systems (OODB)*. On the other hand, researchers and practitioners in the software engineering domain have applied and sometimes adapted existing database technology as a potential solution to their own problems, admittedly not always to their complete satisfaction [e.g., 23, 38].

This paper provides a discussion of the specific database-related requirements of applications from the software engineering domain. Of particular importance is the integration of database functionality in *software engineering environments (SEE)* and its use by software development tools. SEE have been developed for the support and control of various aspects of the software development lifecycle and for the integration of development tools. Finally, current challenges and unresolved issues, as well as future trends and opportunities are considered.

## 2 WHAT IS DATABASE TECHNOLOGY AFTER ALL?

A *database* is a collection of interrelated data, i.e., known facts that can be recorded. A database has the following properties:

- It represents some aspects of the real world called the *miniworld*. Changes to the miniworld have to be reflected in the database.
- It is a logically coherent and integrated collection of potentially large amounts of data with some implicit meaning. This data underlies an explicitly controlled lifecycle; quality properties including consistency, integrity, and security have to be maintained.
- It is designed, built, and populated with—ideally—non-redundant data. It has an intended group of users and some preconceived applications.
- Access to data is provided in user-friendly, flexible, and efficient ways.
- The data can be used and modified independently of the applications that created it. This property is called *data independence*.

In software engineering, the relevant real-world aspects to which the data pertains are software artifacts, software processes, test data, etc. The lifecycle of the data corresponds to the lifecycle of the developed software. The database applications are software engineering tools, which can concurrently access this data.

*Database technology* comprises the concepts, systems, methods, and tools that support the construction and operation of databases, including the following:

**Data modeling.** The concepts available for the definition of mini-worlds are provided by a *data model*. Different data models have varying degrees of expressiveness. The most prominent general-purpose data models are the relational, object-oriented, and object-relational. Special-purpose data models have been developed to support the requirements of specific application domains. Mechanisms are provided to define the structure of stored data (*data definition languages—DDL*). These representation mechanisms are applied to define application-specific *database schemas*[1]. More recently, the proliferation of data on the Internet has motivated the development of modeling concepts for semi-structured or self-describing data.

**Persistence.** The database is stored persistently; i.e., its life span typically extends beyond that of the execution of a particular database application and thus can be used for retrieval and manipulation later.

**Data integrity.** Semantic integrity control mechanisms, such as database triggers, are provided in order to ensure the consistency of the stored data during its manipulation. Consistency constraints may refer to the conformity of implementations to specifications, to the correctness of configurations, etc. The automatic detection of constraint violations and the initiation of repair operations on stored data are also supported.

**Querying.** Stored data may be retrieved according to user-defined criteria. It is an essential requirement that the querying interfaces are flexible and powerful, enabling, for example, declarative query specifications as well as navigation along relationships among stored data. The expressiveness of the query language depends, of course, on the data model used to describe the data. Various standard query languages have been defined including *Structured Query Language (SQL*) [41] for relational databases, and *Object Query Language (OQL)* [14] for object-oriented databases.

**Concurrency control.** A consistent database state must be presented to each user even when concurrent access and manipulation takes place. For this purpose, a transaction model is implemented that allows multiple users to access the database concurrently, and that ensures that a consistent state can be recovered in case of media failure, system crash, or transaction abort. Most widely supported is the *ACID* transaction model [30]. It ensures that database transactions are atomic, preserve a consistent database state, run (logically) isolated from other transactions and have durable effects on the database. The ACID-model fits well business-oriented applications, but is not suitable for long, cooperative design transactions and does not support data sharing and failure recovery for tasks consisting of multiple steps. As a consequence, various extensions have been proposed, collectively known as *advanced transaction models (ATM)*.

**Security.** Security mechanisms are required to prevent malicious or even accidental access by unauthorized users. Several forms of authorization may be granted to a user for accessing specific parts of the database and for modifying the database schema.

**Secondary storage management.** Usually, the amounts of data are too large to fit into main memory. In such cases, database technology provides various techniques such as indexing, clustering, and resource allocation to efficiently manage secondary storage media.

**Compilation and optimization.** A central assumption in databases is the separation of the logical definition of the data from its underlying physical implementation. Translation mechanisms between the applications and the external and logical levels are provided. Whenever possible, changes to the physical data representation must not affect the application logic.

**Distribution and heterogeneity.** In distributed databases, data is stored on several physical locations. In database federations, databases of various degrees of heterogeneity are integrated by a common access layer. Depending on the application, different degrees of distribu-

[1] In the database community the terms data model and database schema are commonly used to denote the conceptual representation mechanisms vs. the result of applying these mechanisms. They are called *metamodel* and *model* respectively in the more accurate terminology used in software engineering.

tion transparency may be required. The controlled replication of data may be used to increase efficiency.

Database functionality can be provided by different types of software systems. *Database management systems* (DBMS) are general-purpose software systems specifically designed to manage large quantities of data in a consistent manner. Their primary goal is to provide comfortable and efficient access to databases. A DBMS implements a specific data model, and provides all or most of the functionality described in the previous paragraphs. A *database system* comprises the database and the database management software. Special-purpose systems, which provide some sort of database functionality in application-specific contexts like SEE, have also been developed.

While in the past database systems have been large, monolithic software systems, a research trend has emerged in recent years towards their componentization. This trend is necessary in order to efficiently provide database functionality within today's distributed object and component architectures, such as, CORBA [46] which defines—among others—interfaces to persistence, transaction, and relationship services. The componentization subsequently allows the flexible bundling of database functionality in a more application-focused way [27]. Bundling is already supported, at least in a simple form, in modern DBMS such as DB2 (extenders, [33]) and Oracle 8 (cartridges, [47]). Examples of more advanced prototypical database component architectures include KIDS [26] and Open OODB [11].

## 3 REQUIREMENTS FOR DATABASE SUPPORT IN SOFTWARE ENGINEERING ENVIRONMENTS

SEE can advantageously use database functionality to create value-added software development services. In this case, databases are used as the data integration medium between the various applications/subsystems of the SEE. For this purpose, a *domain-specific information model* is defined which is then implemented on top of a regular DBMS resulting in a software engineering *repository* [10]. The functional and operational requirements for the efficient and effectiv use of database technology in SEE are discussed in this section.

**Representation and persistent storage of software artifacts.** The issues that have to be solved with respect to representation of software artifacts are determined by the kinds of stored artifacts and the ways these artifacts are used in a SEE. Different kinds of software artifacts, with large variations in size, are created and used during the various software lifecycle phases. These include, for example, analysis and design specifications and diagrams, programming language code, deployment (executable) components, test data, and human-readable documentation. The artifacts have certain metalevel properties as well as a content, and can be interrelated in various ways, e.g., by generalization, aggregation, and *uses*-relationship. However, artifact attribute and relationship types cannot, in general, be predetermined and fixed at type/schema definition time. These observations lead to requirements that affect both aspects of the data model and of the concrete SEE repository schemas. Thus, the database data model must be powerful enough to allow the description of any potentially required artifact type, and the defined persistent artifact descriptions (i.e., the database schema) should be able to dynamically evolve [4] during the lifetime of the software repository.

**Artifact integrity and consistency.** An important issue in software repositories is the maintenance of integrity and consistency of the artifacts with respect to state validity (i.e., static constraints) as well as allowed state transitions, usage context, and development history over time (i.e., dynamic constraints).

**Artifact querying and retrieval.** The efficient retrieval of software artifacts from software libraries is a prerequisite for software reuse. The techniques that can be used for retrieval depend on the used artifact modeling techniques and representation mechanisms. In any case, it is important to achieve a high retrieval *precision* (i.e., a high ratio of relevant artifacts among those retrieved), a high *recall* ratio (i.e., a high retrieval ratio with respect to the number of the relevant artifacts in the database), and a short *response time* (i.e., a high inspection rate with respect to the database size).

**Version and configuration management.** During its lifetime, a software artifact evolves, reflecting its state and progress of development. In order to effectively support software development, various *versions* of this state as well as the relationships between these versions must be stored. Furthermore, complex software systems are typically aggregated from a specific set of versions of smaller components resulting in specific *configurations*. It is common practice in SEE to integrate a dedicated software configuration management system (SCMS), which allow the control of evolving software systems. While traditional SCMS such as RCS [52] provide basic versioning and composition functionality, they are typically file-based and support the versioning of large granularity artifacts, such as files. In various development scenarios, as for example, in class-based reuse, this capability is not adequate, as the granules of versioning and composition do not correspond to individual files. When customized version management granularity is required, an appropriate version model must be supported by the artifact storage system.

**Tool integration and portability.** The integration of the tools used for the development of software is an important requirement in SEE. Data (i.e., software artifacts) produced by one tool should be directly usable by other tools in the environment[2]. A complementary issue is sup-

[2] The control and presentation integration of tools are further important requirements in software engineering

port for the portability of tools across different SEE. Data integration can be achieved by providing a common artifact storage infrastructure for environment tools. Tool portability is facilitated if access to the storage infrastructure is through well-defined interfaces. Both aspects can be supported by appropriate repository services and open service interfaces.

**Support for cooperative design transactions and workspace management.** In software development practice it is necessary to manage work-in-progress and consider the nature of the performed work, which typically consists of long-duration cooperative, design transactions. This is typically supported by providing private, persistent workspaces within which users manipulate and store private pieces of work before they make them visible and accessible to the global pool of developers in their team. Typically, work items are checked-out to a private workspace and a new version is created by the subsequent check-in operation. Furthermore, specialized transaction mechanisms are required which relax isolation and allow the exposure and exchange of intermediate work products among developers in a group.

**Representation and enactment of software processes.** A *software process* comprises a partially ordered set of process steps intended to reach a goal related to software development. Process steps can be either composite subprocesses or elementary activities. A process step can have resources allocated to it, which are required for the it to be performed. A process model is a more or less formal abstract description of a software process [40]. The level of abstraction of a process model determines the details of definition and implementation that are described. In any case, various aspects of the software process have to be modelled including the types of artifacts being manipulated, the step types and structure, the resources used, and the organization within which the process is performed. The process models can subsequently be enacted by an appropriate engine. During the enactment, instances of the model entities, i.e., process execution data, are created and manipulated. This data can be managed by a database system, which may in addition provide basic enactment functionality.

**Collection and evaluation of software development metrics.** During recent years, the importance of instrumenting and measuring the software development process has been recognized. Software metrics can be used to efficiently guide the software lifecycle and improve the development process [48]. For this purpose, it is important that process enactment data are collected and persistently stored for subsequent evaluation.

environments. Although database functionality can be applied towards their achievement, the main issue considered in this paper is that of data integration.

## 4 DATABASE TECHNOLOGY FOR SOFTWARE ENGINEERING: EXISTING SOLUTIONS

In this section, we survey in what ways and to which extent the requirements described in the previous section have been met by database technology so far.

**Defining software engineering repositories.** The successful construction of a software repository depends largely on the data model implemented by the underlying database system. While the relational data model is generally not considered powerful enough for the modeling of software repositories, the object-oriented and more recently the object-relational data model have been successfully used for this purpose. Special-purpose data models have also been developed (e.g., the graph-based model in GRAS [39]). The evolution of a software repository can be supported by metalevel mechanisms (i.e., metadata and meta-object protocols as implemented in some object-oriented DBMS) which operate on the types of the stored artifacts.

Probably the most important and comprehensive standardization effort in defining a software repository is the Portable Common Tool Environment PCTE [53]. PCTE defines a set of open interfaces that can be used by software engineering tools thus, enabling their seamless data, presentation, and control integration. The interfaces provide

- object lifecycle management,
- schema management,
- program execution and communication,
- concurrency and integrity control,
- security policy enforcement and auditing,
- physical data distribution management, and
- resource accounting.

A PCTE-compliant repository supports a graph-based data model that can be used to describe typed design, implementation, execution and documentation objects, as well as the relationships (links) between them. Despite its functionality, PCTE has only been used in relatively few SEE (e.g., ALF [12]) as it is hampered by a number of important deficiencies, including the lack of support for object-oriented modeling and triggers. A number of extensions to PCTE have been developed, such as H-PCTE [37] and P-RooT [13] which overcome these limitations.

**Cooperative design transactions.** The limitations of ACID transactions in the support of long-living units of work, user control of transaction execution, and synergistic cooperation [6] have led to the development of advanced transaction models (discussed in, e.g., [35, 28]). The precursor to all these are without doubt *nested transactions* [43] which extend the single-level transaction structure, provide a mechanism for fine-tuning the scope of rollback, and allow composite units of work

to be reflected as transaction structures. Strict isolation is relaxed, e.g., in *open nested transactions* (e.g., Sagas [25]) which allow results of subtransactions to be committed independently from the parent transaction. In order to avoid cascading aborts in case a nested transaction fails, compensating transactions must be known for each subtransaction. A survey on these issues and mechanisms can be found in [6].

Despite the plethora of ATM, only few have actually been implemented in DBMS let alone in commercial ones. Some systems, such as Damokles [21], Objectstore, and Versant support design transactions by workspaces and check-out/check-in mechanisms. The Coo nested open transaction model has been implemented over the P-RooT object-oriented extension of PCTE [13]. A pragmatic approach is presented by [3] which describe a technique for the implementation of work-in-progress by appropriate extensions to the schema of the repository and the automatic modification of query and update operations against this new schema.

**Artifact integrity and consistency.** Various mechanisms and technologies have been developed for integrity control in database systems. In relational database systems, static consistency constraints can be defined to a certain extent within the database schema, for example, through attribute domains, primary keys, referential integrity constraints, and assertions in the latest edition of the popular SQL-standard, SQL:1999 [22]. OODB currently provide type checking mechanisms. DBMS implement these constraints, for example, with indexes, type-checking mechanisms, and triggers as provided by active database systems [55]. Integrity control mechanisms can be advantageously used on software engineering artifacts to automate manual consistency checks, provide information about tool-provoked errors, and simplify the logic implemented in software tools.

**Artifact querying and retrieval.** While practically all current DBMS provide a boolean query model with all-or-nothing semantics, information retrieval techniques have been developed for probabilistic queries, e.g., based on similarity metrics. Such techniques are currently being provided as functional extensions of mainstream object-relational database systems and could be advantageously used in software engineering repositories. Furthermore, the development of query languages for semi-structured (self-describing) data is currently a hot research issue [2]. They allow to query, among others, web-based data sources.

**Support for software process modeling, enactment, and evaluation.** The modeling of software processes can be implemented within a software engineering repository. The explicit representation of software process elements in a SEE repository, as advocated *by process-centered software engineering environments (PCSEE)* [24], provides a series of advantages enabling process evolution and reasoning about process properties. In this context, database functionality can be used both for defining and managing process models, as well as for environment notification and process enactment, e.g., by active database mechanisms. A large amount of research on similar issues has been done in the area of workflow management—following the seminal work by [20]. Despite the fact that there are some differences in the nature of software processes and workflows, also significant similarities make much of the technology applicable in the software engineering domain. This is supported by the fact, that some prototypical software process environments have evolved to also support workflow execution (e.g., Oz [9]).

The storage and analysis of software process-related metrics can be efficiently implemented with a series of mature database technologies. In this context, the use of database systems is straightforward and poses similar requirements to those of database applications in business data processing. The special requirements stem from the fact that data collection should be non-intrusive with respect to the enactment of the software process. This depends on the degree of integration of the database systems with the SEE. Especially in PCSEE which are built around a general-purpose database engine (e.g., SPADE-1 [5]) in which process information is explicitly stored, this may be achieved through appropriate schema extensions and query definitions.

## 5 SOFTWARE ENGINEERING DATABASES: PAST AND PRESENT

The use of database functionality in SEE and CASE tools has a story spanning the last two decades. Quickly reaching the limitations of file-based storage and experiencing the shortcomings of contemporary mainstream database technology, various special-purpose systems have been developed to satisfy requirements that were not covered by those. Their principal extensions relate to the supported data model, versioning, and transaction models. In some cases, as for example concerning the support for object-oriented data modeling, the technology developed for SEE (as well as other engineering) applications has influenced to a large extent current advanced general-purpose database systems like object-oriented and object-relational ones. In other cases, however, mainly with respect to extending the classical ACID transaction model, their practical impact has been minimal. Furthermore, the use of proprietary technology has often hampered the commercialization and large-scale use of SEE.

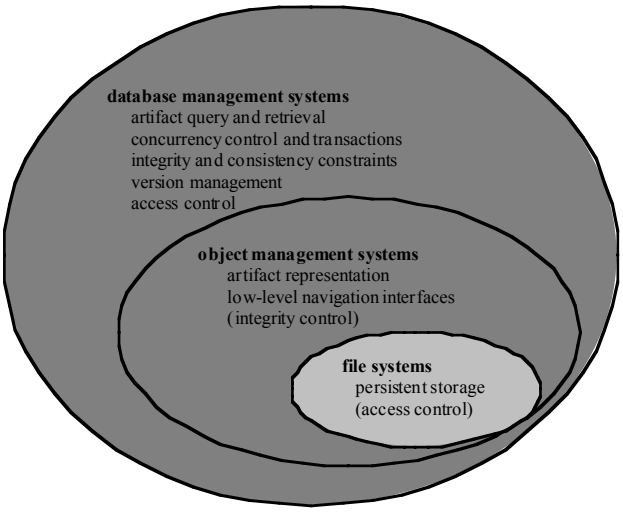


Fig. 1: The relationship between file system, OMS and DBMS functionality.

**DBMS or OMS?**

The use of *object management systems* (OMS) which provide database functionality in SEE has been widespread in the software engineering community. Although the distinction between DBMS and OMS is not sharp, typically OMS provide artifact representation and persistence, rudimentary navigation interfaces, and low-level integrity control mechanisms (see also Fig. 1) only. The principal reason for the implementation and use of OMS has been the perceived or real limitations of available DBMS combined with the complexity of implementing a full-fledged DBMS.

A number of prototypical OMS have been built from the mid 80's to the early 90's (e.g., PLEIADES [51], Adele-DB [7], and Triton [31]) and used in different SEE (see also Table 1).

**Proprietary Software Engineering Databases**

A number of proprietary DBMS have been built for SEE. Three main limitations of contemporary DBMS were addressed in these systems: modeling of software artifacts and their relations, version management, and support for cooperative transactions.

Two significant research prototypes were EPOSDB and GRAS. The structurally object-oriented EPOSDB [17], used in the EPOS SEE, is built upon the C-ISAM sequential file system and provides a change-oriented versioning model and nested, cooperative transactions. The GRAS DBMS [39], used in IPSEN, provides a graph-based data model in which software artifacts are represented as attributed nodes and artifact relationships as arcs connecting these nodes. A main incentive for using this data model has been the support for artifact classification and reuse. A series of other important software engineering database systems (for a non-exhaustive list see Table 1) were developed; their description and evaluation, however, is beyond the scope of this paper.

**Commercial DBMS for SEE**

While purpose-built software engineering database systems have been developed in the past, there is a more recent tendency, mainly due to the richer functionality of modern DBMS, to use or extend off-the-shelve systems. Whenever off-the-shelve systems can be used, mature database functionality is available at a relatively low cost.

Commercial relational DBMS have been used in various prototypical SEE as component repositories. Sybase, for example, is used to store artifact descriptions in JBCL [36]. Commercially available object-oriented DBMS have also been used, either as-are, e.g., $O_2$ in Memphis [54], in Merlin to store process and organizational data, and has been extended with additional functionality as, e.g., NAOS [16] in GOODSTEP environments. An object-relational DBMS-based software artifact repository is described in [49]. It is used for the management of software process experience data as well as for controlling the access of tools to design data.

Some current commercial CASE tools use OODB for (partial) data integration. Examples include the Object Engineering Workbench [34], instances of which may use Poet or Objectstore to persistently store and manage modeling artifacts and project management data, and MetaEdit+ [42] built around the Smalltalk-based OODB ArtBase.

Table 1: Some special-purpose software engineering database and object management systems.

| Name | Type of system | Used in SEE | Reference |
|---|---|---|---|
| Adele-DB | Structurally object-oriented OMS | Adele 2 | [7] |
| ALF OMS | PCTE-based repository | ALF | [12] |
| Cactis | Semantic DBMS | - | [32] |
| DAMOKLES | Structurally object-oriented DBMS | UNIBASE | [21] |
| EPOSDB | Structurally object-oriented DBMS | EPOS | [17] |
| GRAS | Graph-based active DBMS | IPSEN, Melmac | [39] |
| H-PCTE | PCTE-based repository | - | [37] |
| Marvel OMS | Structurally object-oriented OMS | Marvel | [8] |
| NAOS | Active layer on top of $O_2$ object-oriented DBMS | GOODSTEP | [16] |
| Pleiades | Database programming language | Arcadia | [51] |
| P-RooT | PCTE-based repository | Coo | [13] |
| SIB | Proprietary repository | ITHACA | [19] |
| Triton | Extension of the Exodus OMS | Arcadia | [31] |

## 6 DATABASE TECHNOLOGY FOR SOFTWARE ENGINEERING: THE CHALLENGES AHEAD

The current challenges to the database community stem from both the advancing requirements in the software development domain and inadequacies of current database technology.

**Separation of concerns**. As in any other application domain, database support for software engineering applications can be located somewhere on a spectrum from "fat" to "thin" servers. In the first case, one tries to handle most tasks in the underlying database system, and very little remains to be done in applications (i.e., the software engineering tools). On the other extreme, only rudimentary support is given, e.g., in the form of an OMS, and most of the "intelligence" is in applications and tools. Both extremes, of course, come with different advantages and drawbacks, e.g., with respect to what remains to be done in applications and the degree of flexibility tools and applications are provided with.

If this separation of tasks between DBMS and software engineering applications is not addressed properly, then either software engineering applications do not receive sufficient support, or they receive support in the wrong and therefore less useful (or even useless) form, or finally there is duplication of work among the applications and the database. As an example, consider some of the advanced transaction models that attempt to control on a very fine-grained level interaction and cooperation of designers in design applications. Alternatively, the same task might be addressed by process models defined on top of the database system (yet with appropriate support for defining, enacting, and monitoring processes). Thus, before questions of how and in which form to support software engineering applications of database systems can be adequately answered, the issues of separation of concerns between them and of the definition of a proper architecture has to be addressed.

**Database technology for software engineering metasystems.** Software has always been a non-tangible product. Consequently, software development can inherently involve widely dispersed teams, often composed across different organizations. This is even desirable in order to optimize the use of available resources and minimize development costs. Database technology should provide the infrastructure to integrate software engineering teams through a metasystem [29] that gives users the illusion of a large transparent computational environment. In such a metasystem, software engineers can share artifacts, computational objects, and tools in a common virtual workspace without being aware of physical program execution or artifact storage locations. From a database perspective, mechanisms to support a shared persistent object space, heterogeneous querying, intelligent object relocation and caching, fault tolerance, and security must be provided.

**Querying heterogeneous data sources.** The efficient reuse of software artifacts of varying granularity in a software engineering metasystem depends on effective classification, search and retrieval techniques. It is important that heterogeneous artifact types spanning the entire spectrum from text to semi-structured data (e.g., HTML, XML) to graphical representations to deployment components can be retrieved through similar interfaces. To this end, intelligent query processing and powerful meta-information management is needed. The query language must have certain properties, such as being expressive and allowing restructuring of data, having precise semantics, and support compositionality (i.e., the output of one query should be usable as input of another) [2]. The development of such query languages is currently an area of intensive research.

**Support for flexible cooperation schemes and creative work.** Research in ATM has not (yet) led to implementations that are feasible for a broad range of scenarios. In the current state of the art, it seems that the task of enabling and controlling collaborative work is better supported by process-centered environments and workflow management systems. The basic open problem in this context, therefore, is how these environments can benefit from advanced database technology. In particular, it must be determined how to leverage integral elements of the transaction concept such as consistency control and recovery to software engineering (environments).

A consistency checking service can ensure that all artifacts created within a software process are consistent with respect to the underlying constraints. The definition of such a service is particularly difficult whenever artifacts are stored in a heterogeneous collection of systems, and are not under the control of a single DBMS. A failure handling service can provide recovery from inconsistent states such as the impossibility to correctly complete a design task. Adequate techniques to accomplish this task range from dynamically modifying the enacted process to compensation to exception handling techniques. Anyhow, a lesson that can be drawn from previous experiences in ATM-research is that finding generally applicable yet adequate solutions is a true challenge.

**Unbundling and rebundling database components.** The proper abstractions and mechanisms for providing flexible database support to software engineering applications and metasystems are yet to be defined. It is important to be able to fine-tune and provide the functionality required at each point of a software engineering metasystem without imposing overhead. To achieve this, we postulate that component technology has to be applied, in which functional components providing customized database services are deployed and managed in a requirement-driven manner.

The proposed approach has as its core ingredients the terms "unbundling" and "rebundling" [27]. Unbundling refers to the decomposition of a class of systems into a set of reusable components and identified relationships between them. Subsequently, rebundling allows the implementation of customized database services by components constructed in a systematic way. Rebundling requires an appropriate architectural framework, a well-defined bundling process and supporting environment, a repository of available reusable "bundable" components providing identified database services, and appropriate mechanisms for plugging these components together. This may allow the optimized deployment of "lightweight" database servers at exactly those points within a software engineering metasystem where the functionality is required.

## 7 CONCLUSIONS

Modern database technology provides a rich set of functionality that is able to meet several requirements of software engineering applications in a satisfactory manner. However, the successful integration of the two domains has not yet occurred. We believe that this is—largely—because research and development in the two communities has rarely been coordinated. It is thus important that the interdisciplinary work is encouraged and common forums are established allowing a constructive exchange of ideas and experience.

## 8 REFERENCES


1. Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995.
2. Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.
3. Ahmed, R. and Dayal, U. Management of Work in Progress in Relational Information Systems. In *Proc. 3rd CoopIS,* 1998.
4. Banerjee, J., Kim, W., Kim, H.-J., and Korth, H. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *Proc. ACM SIGMOD Conf.,* 1987.
5. Bandinelli, S., Di Nitto, E., and Fuggetta, A. Supporting Cooperation in the SPADE-1 Environment. *IEEE Transactions on Software Engineering* 22, 12 (December 1996), 841-865.
6. Barghouti, N. and Kaiser, G. Concurrency Control in Advanced Database Applications. *ACM Computing Surveys* 23, 3 (September 1991), 269-317.
7. Belkhatir, N., Estublier, J., and Melo, W. ADELE-TEMPO: An Environment to Support Process Modelling and Enaction. In J.-C. Derniam, A. Finkelstein, J. Kramer, B. Nusebeih, Eds., *Software Processing Modelling and Technology*, 1994.
8. Ben-Shaul, L., Kaiser, G., and Heineman, G. An Architecture for Multi-User Software Development Environments. In *Proc. ACM SIGSOFT '92*, 1992.
9. Ben-Shaul, I., and Kaiser, G. Integrating Groupware Activities into Workflow Management Systems. In

*Proc. 7th Israeli Conf. on Computer Based Systems and Software Engineering*, 1996.
10. Bernstein, P. Repositories and Object-Oriented Databases. *SIGMOD Record* 27, 1 (March 1998), 88-96.
11. Blakeley, J. Open OODB Architecture and Query Processing Overview. In A. Dogaç, T. Özsu, A. Billiris, T. Sellis, Eds., *Advances in Object-Oriented Database Systems*. Springer, 1994.
12. Canals, G., Boudjlida, N., Derniame, J.-C., and Lonchamp, J. ALF: A Framework for Building Process-Centered Software Engineering Environments. In J.-C. Derniam, A. Finkelstein, J. Kramer, B. Nusebeih, Eds., *Software Process Modeling and Technology*, 1994.
13. Canals, G., Charoy, F., Godart, C., and Molli, P. P-RooT & Coo: Building a Cooperative Software Development Environment. In *Proc. 7th Conf. on Software Engineering Environments (SEE' 95)*, 1995.
14. Cattel, R. et al. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.
15. Ceri, S. and Widom, J.. Deriving Production Rules for Constraint Maintenance. In *Proc. 16th VLDB Conf*, 1990.
16. Collet, C. and Coupaye, T. The NAOS System. In *Proc. ACM SIGMOD Conf.*, 1995.
17. Conradi, R., Liu, C., and Hagaseth, M. Planning Support for Cooperating Transactions in EPOS. *Information Systems* 20, 4 (1995), 317-336.
18. Conradi, R., Ed. *Proc. Software Configuration Management 7 Workshop*, 1997.
19. Constantopoulos, P., Jarke, M., Mylopoulos, M., and Vassiliou, Y. The Software Information Base: A Server for Reuse. *VLDB Journal* 4, 1 (1995), 1-43.
20. Dayal, U., Hsu, M., and Ladin, R. Organizing Long-Running Activities with Triggers and Transactions. In *Proc. ACM SIGMOD Conf.,* 1990.
21. Dittrich, K.R., Gotthard, W., and Lockemann, P.C. DAMOKLES: A Database System for Software Engineering Environments. In R. Conradi, T. Didriksen, D. Wanvik, Eds., *Advanced Programming Environments*. Springer, LNCS 244, 1986.
22. Eisenberg, A. and Melton, J. SQL: 1999, formerly known as SQL3. *ACM SIGMOD Record* 28, 1 (1999), 131-138.
23. Emmerich, W., Schäfer, W., and Welsh, J. Databases for Software Engineering Environments—The Goal has not yet been attained. In *Proc. European Software Engineering Conf.*, 1993.
24. Fuggetta, A. and Wolf, A., Eds. *Software Process*. John Wiley & Sons, 1996.
25. Garcia-Molina, H. and Salem, K. Sagas. In *Proc. ACM SIGMOD Conf.*, 1987.
26. Geppert, A. and Dittrich, K.R. Strategies and Techniques: Reusable Artifacts for the Construction of Database Management Systems. In *Proc. 7th Int'l Conf. on Advanced Information Systems Engineering*, 1995.
27. Geppert, A. and Dittrich, K.R. Bundling: Towards a New Construction Paradigm for Persistent Systems. *Networking and Information Systems Journal* 1, 1 (1998), 69-102.
28. Gray, J. and Reuter, A. *Transaction Processing Concepts and Techniques*. Morgan Kaufmann, 1993.
29. Grimshaw, A., Ferrari, A., Lindahl, G., and Holcomb, K. Metasystems. *Communications of the ACM* 41, 11 (1998), 46-55.
30. Haerder, T. and A. Reuter, A. Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys* 15, 4 (December 1983), 287-317.
31. Heimbinger, D. Experiences with an Object Manager for a Process-Centered Environment. In *Proc. 18th VLDB Conf.*, 1992.
32. Hudson, S. and King, R. The Cactis Project: Database Support for Software Environments. *IEEE Transactions on Software Engineering* 14, 6 (June 1988).
33. IBM Corp. *DB2 Relational Extenders,* 1995.
34. Innovative Software. Object Engineering Workbench Product Overview. http://world.isg.de/, December 1999.
35. Jajodia, S. and Kerschberg, L., Eds. *Advanced Transaction Models and Architectures*. Kluwer Academic Publishers, 1997.
36. Keqin, L., Lifeng, G., and Fuqing, Y. An Overview of JB Component Library System. In *Proc. TOOLS-24*, 1997.
37. Kelter, U. H-PCTE–A High Performance Object Management System for System Development Environments. In *Proc. COMPSAC'92,* 1992.
38. Kelter, U. and Däberitz, D. An Assessment of Non-Standard DBMSs for CASE Environments. In *Advances in Database Technology–EDBT'96*. Springer, LNCS 1057, 1996.
39. Kiesel, N., Schürr, A., and Westfechtel, B. GRAS: A Graph-Oriented Software Engineering Database System. In M. Nagl, Ed., *Building Tightly Integrated Software Development Environments: The IPSEN Approach*. Springer, LNCS 1170, 1996.
40. Lonchamp, J. A Structured Conceptual and Terminological Framework for Software Process Engineering. In *Proc. 2nd Int'l Conf. on the Software Process - Continuous Software Process Improvement*, 1993.
41. Melton, J. and Simon, A. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann, 1993
42. Metacase. MetaEdit+ Product Overview. http://www.metacase.com/, November 1999.
43. Moss, J.E.B.. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, 1985.
44. Nagl, M., Ed., *Building Tightly Integrated Software Development Environments: The IPSEN Approach*. Springer, LNCS 1170, 1996.

45. Nodine, M.H., Ramaswamy, S. and Zdonik, S.B. A Cooperative Transaction Model For Design Databases. In A.K. Elmagarmid, Ed. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.
46. The Object Management Group. *The Common Object Request Broker Architecture and Specification, Revision 2.0*, 1995.
47. Oracle Corporation. *Managing Text with the Oracle 8 ConText Cartridge*. Oracle Technical White Paper, 1997.
48. Paul, R., Kunii, T.L., Shinagawa, Y., and Khan, M. Software Metrics Knowledge and Databases for Project Management. *IEEE Transactions on Knowledge and Data Engineering* 11, 1 (1999).
49. Ritter, N., Steiert, H.-P., Mahnke, W., and Feldmann, R. An Object-Relational SE-Repository with Generated Services. In *Proc. Int'l Resource Management Association Conf. 1999, Managing Information Technology Resources in Organizations in the Next Millenium (Computer-Aided Software Engineering Track)*. IDEA Group Publications, 1999.
50. Reiss, S.P. Simplifying Data Integration: The Design of the Desert Software Development Environment. In *Proc. 18th Int'l Conf. on Software Engineering*, 1996.
51. Tarr, P. and Clarke, L. PLEIADES: An Object Management System for Software Engineering Environments. In *Proc. of the ACM SIGSOFT '93 Symposium on the Foundations of Software Engineering*, December 1993.
52. Tichy, W. RCS – A System for Version Control. *Software Practice and Experience* 15, 7 (July 1985) 637-654.
53. Wakeman, L. and Jowett, J. *PCTE: The Standard for Open Repositories. Foundation for Software Engineering Environments.* Prentice Hall, 1993.
54. Werner, C.M.L., Travassos, G.H., da Rocha, A.R.C., de Cima, A.M. , da Silva, M.F., and de Vasconcelos, F.M. Memphis: A Reuse-Based Software Development Environment. *Proc. TOOLS-24*, 1997.
55. Widom, J. and Ceri, S., Eds. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996.