A Formal View Integration Method

Joachim Biskup and Bernhard Convent

Universitat Dortmund
Informatik VI
Postfach 500 500
4600 Dortmund 50
Federal Republic of Germany

Abstract The design of an appropriate conceptual database scheme is one of the most difficult tasks in usual database applications Especially, the design of a common global database scheme for many different user groups requires a great amount of effort and skill, because the desired scheme should fit a great variety of requirements and expectations Here, view integration is a natural method that should help to manage the complexity of such a design problem For each user group the requirements and expectations are separately collected and specified as views, that are subsequently integrated into a global scheme supporting all those different views
In this paper, we carefully develop a formal model, clarifying many notions and concepts, related to the view integration method This formal model serves as a theoretical basis of our integration approach that uses equivalence preserving, local scheme transformations as the main integration operations

1 INTRODUCTION

View integration is a method that should help to cope with problems related to the design of conceptual database schemes for extensive database applications In such an extensive application, there exist a lot of different user groups, each of them requiring particular information on the whole application to fulfill their special tasks I e , each user group has its own view of the whole application which can formally be described by a small (view) database scheme, representing the information needs and rights, associated with that particular user group

Of course, one of the main goals in database applications is to centralize all information in a single global database that can support all views By this centralization, redundancy is avoided and the threat of inconsistancies between the views is reduced So, several database schemes for the different views and a single, global database scheme for the centralized global database have to be designed

According to a common approach, first the global database scheme is designed, and afterwards suitable portions of it are selected to form the view schemes for the different user groups But this approach bears at least the following deficiencies

- The immense number of expectations and requirements of all user groups can hardly be managed and overviewed Thus, it is rather difficult to get a suitable' global scheme that really can fit all those requirements
- The classical formal design algorithms ([Codd2] [Fagi], [BDB]) can hardly be used to find a global scheme of an extensive application These algorithms are based on a specification of the form (**U C**), where **U** is a set of attributes (the universe), and **C** is a set of functional and multivalued dependencies Usually, such specification means are not sufficient to capture all the relevant details of an extensive application Thus, the classical approaches are rather inadequate to the design of global schemes, while they are still quite useful, when designing smaller user views

So, view integration seems to be a much more natural approach At first, the requirements and expectations of all user groups are separately collected and specified as view schemes Thereafter, all those views are integrated into a global scheme that supports all views

This view integration approach can be embedded into a methodology for the whole database design process According to [Lum], [TeFr], [YNW] the design process can be divided into five phases

Requirement Analysis The requirements of all user groups are analyzed, collected and formally specified so that both the information and processing requirements are made explicit

View Modelling Using the information requirements of the first phase, a database scheme for each user group is constructed, representing its special view of the application These user views are generally specified on a more abstract level of a conceptual data model, which should be independent of any concrete database management system

This research was supported in part by the Deutsche Forschungsgemeinschaft under grant number Bi 311/1-1

View Integration All the user views are analyzed in order to specify the connections and relationships between them and to detect possible conflicts Then the views are integrated into a global scheme for the centralized, common database that should be able to support all views This global scheme is also specified on the abstract level of the chosen data model

Implementation Design The global conceptual database scheme is refined in order to get a processible scheme of the target database management system

Physical Design On the physical level of the target database management system, an appropriate storage structure is chosen to get an efficient implementation of the whole application

The above design phases are still of such a size and complexity that any of them can hardly be done without interactive computer support Thus, there is a need for computer-aided design tools for all phases of the whole design process In order to be useful and acceptable, such tools should be based on a clear, formal background Although several work ([BaLe,BLM], [CaV1] [MaW1,W1Ma], [NaGa], [YWH]) was directed to the view integration phase, such a complete formal basis is still missing At least the following details should be clarified as a basis for a computer-aided view integration tool

1) A data model with a clear formal semantics must be chosen This data model forms a kind of interface to the other phases, since all user views and the global scheme are specified within this model
2) A formal specification language must be given that can be used to describe the connections and relationships of different views
3) The notion of conflicts between several views has to be formalized
4) There is a need for a formal concept of the inclusion of database schemes, describing when a global database scheme supports a user view
5) Using the above concepts, a complete formal description of the view integration problem should be elaborated

In our research, we try to develop such a theoretically based view integration method, extending many concepts and results of a former approach due to Casanova/Vidal [CaV1]

The paper is organized as follows In Section 2 we carefully develop our formal model clarifying the notions and concepts, needed for the view integration phase In Section 3, our integration algorithm is presented that uses equivalence preserving local scheme transformations as the main integration operations Section 4 concludes this paper by pointing out some open problems and directions for further research

2 A FORMAL MODEL FOR THE VIEW INTEGRATION

2 1 PRELIMINARIES

At first, we briefly present some basic definitions and notation, used throughout the paper

Let **U** be a fixed finite set of attributes and let **T** be a fixed finite set of basic types i e each T∈**T** is a countable set of values containing at least 2 elements A function dom **U**-->**T** associates each attribute with a basic type An ordered subset X={A1, An}⊂**U** will be represented as X=A1 An i e as a word over **U** For two ordered subsets X, Y⊂**U** the assertions X⊂Y, X∩Y=ϕ will hold if the same assertions hold for the corresponding unordered sets Furthermore, the function dom is extended to ordered subsets of **U** by the unique morphism dom **U*** --> **T***

A ***relation scheme*** R[X] consists of the relation name R and an ordered set X=A1 An⊂**U** of attributes A ***tuple t over X*** is an element t∈dom(A1)× ×dom(An) A ***relation r over X*** is a finite set of tuples over X A ***database scheme*** D=<**R**|**C**> consists of the name D, a finite set **R**={R1[X1], ,Rn[Xn]} of relation schemes and a finite set **C** of integrity constraints A ***database d=(r1, ,rn) over R*** associates each relation scheme Ri[Xi] with a relation ri over Xi A database d over **R** that additionally satisfies all integrity constaints in **C** is called an ***instance*** or a ***valid database of D***, denoted by d∈sat(**C**) = $\bigcap_{c \in \mathbf{C}}$sat(c) The ***set of all valid databases of D*** is denoted by I(D)

Integrity constraints are understood as statements about time-invariable properties of the real world that must be satisfied by every meaningful database So, integrity constraints are a formal means to distinguish between meaningful and meaningless databases with respect to the real world Here, we concentrate on functional dependencies (FDs), inclusion dependencies (INDs) and exclusion dependencies (EXDs) A ***functional dependency over R*** [Codd2] is an expression c of the form c≡Ri Y->Z, where Y, Z⊂Xi A database d=(r1, ,rn) over **R** satisfies c, denoted by d∈sat(c), iff for all t, t'∈ri t[Y]=t'[Y] ==> t[Z]=t'[Z] An ***inclusion dependency over R*** [CFP] is an expression c of the form c≡Ri[Y]⊂Rj[Z], where Y⊂Xi, Z⊂Xj and dom(Y)=dom(Z) d∈sat(c) iff ri[Y]⊂rj[Z] Similarly, an ***exclusion dependency c over R*** [CaV1] is of the form c≡Ri[Y]||Rj[Z], where Y⊂Xi, Z⊂Xj, dom(Y)=dom(Z) and d∈sat(c) iff ri[Y]∩rj[Z]=ϕ

Remark Note, we use ***ordered*** sets of attributes instead of usual, unordered sets and we define tuples to be ***sequences*** instead of mappings This notation turns out to be quite useful in connection with inclusion and exclusion dependencies Beyond these basic definitions, we assume the reader to be familiar with the operations of the relational algebra, as for instance described in [Maie] [Ullm]

2 2 THE DATA MODEL

Within conceptual database design a data model is used to develop an abstract representation of the relevant parts of the real world, which should not depend on the database management system chosen for the implementation (See [Brod] for an overview of data models) As already indicated, our analysis of the view integration problem is based on a relational model of data [Codd1]

In the following, we adopt some common restrictions on the use of integrity constraints to get a proper subclass of so-called normalized database schemes This normalization should guarantee that only semantically meaningful integrity constraints are used

Let D=<**R**|**C**> be a database scheme D is called a ***proper database scheme***, iff
for all relation schemes Ri[Xi]∈**R** there is a pair (Ki,Pi), where Ki∩Pi=ϕ KiPi=Xi and
a) for all Rj[Xj]∈**R**
the set of all FDs refering to Rj is equivalent to the single FD Rj Kj->Pj,
b) for all Rj[Y]⊂Ri[Z]∈**C**
Rj≠Ri and ((Y=Kj and (Z⊂Ki or Z⊂Pi)) or (Z=Ki and (Y⊂Kj or Y⊂Pj))),

c) for all $R_j[Y] \| R_l[Z] \in \mathbf{C}$
$R_j \neq R_l$, $Y = K_j$ and $Z = K_l$

a) restricts the use of FDs as intrarelational constraints A proper database scheme is a collection of entities and relationships, each identified by a unique key Attributes in K_i are called ***identifying attributes*** of the objects (entities or relationships) of type R_i, whereas attributes in P_i are called ***property attributes***

b), c) restrict the use of INDs and EXDs as interrelational constraints, guaranteeing that they are always via keys, which make them likely to be supported by concrete database management systems An IND of the form $R_j[K_j] \subset R_l[Z]$ denotes that in any instance $d=(r_1, \ldots r_n)$ all objects of type R_j, occuring in r_j, are used either as an identifying or as a property part of objects in r_l An IND of the form $R_j[Y] \subset R_l[K_l]$ denotes that the properties of any object of type R_l occuring either as identifying or as property part of an object in r_j, can be found in r_l An EXD $R_j[K_j] \| R_l[K_l]$ denotes that the objects in r_j and r_l are distinct, although they belong to a common general type of objects

Although the above restrictions seem to be technically rather complicated we believe that they are usually satisfied when modelling real world situations All views and the desired global scheme should be specified as proper database schemes Thus, it is the task of the view modelling phase to supply all views in that normalized form

We finish this paragraph with a still rather small example that will often be reused throughout the paper

Example Suppose that a small trading company is divided into four divisions (personnel, pay-roll, marketing and stock division) Of course, each division has its own special information needs which will be modelled as a proper database scheme We present these views $V_i = \langle \mathbf{R_i} | \mathbf{C_i} \rangle$ in the usual tableau-oriented manner with the identifying attributes underlined and the corresponding FDs omitted The following abbreviations are used Employee Division, Identity, Salary, Account, Order-date, Delivery-date, Quantity, Product

V1 (personnel division)

Employees	#Emp	Name	Address	Date	Div

V2 (pay-roll division)

Salaries	#Id	Name	Sal	#Acc

Pay-roll-emps	#Id	Address

Pay-roll-emps[#Id] $\subset$ Salaries[#Id]

V3 (marketing division)

Articles	#Article	Name	Price

Customers	#Customer	Name	Address

Orders	#Order	#Customer	Ord-date	Del-date

Order-contents	#Order	#Article	Qty

Marketing-emps	#Emp	Name	Address

Orders[#Order] $\subset$ Order-contents[#Order],
Orders[#Customer] $\subset$ Customers[#Customer],
Order-contents[#Order] $\subset$ Orders[#Order],
Order-contents[#Article] $\subset$ Articles[#Article]

V4 (stock division)

Stock	#Prod	Name	Qty

Stock-emps	#Emp	Name	Address

2 3 INTEGRATION CONSTRAINTS

Let V1, ,Vn be a set of database schemes that have to be integrated into a single global scheme As all these schemes represent overlapping portions of the same part of the real world, normally, a considerable amount of data is shared These connections between the different views must be analysed and specified Statements describing time-invariable connections between several views are called ***integration constraints***

In our approach, we use integration constraints with a formal semantics essentially based on the semantics of INDs and EXDs But nevertheless, we stress that it is important not to mix up integrity constraints and integration constraints, as done in [CaVi], because of different intended meanings Integration constraints describe the connections between different views, and so they form some kind of structural inter-view constraints which have to be considered during integration On the contrary, integrity constraints specify properties of the real world, and so they determine what are meaningful databases Thus, throughout the paper a different syntactical notation is used to distinguish the two types of constraints

In the following, we first present some typical connections between different views using our former example again A complete formal definition of the syntax and semantics of the integration constraints will be given afterwards

Example (continued) Naturally, all employees (and only employees) draw a salary So the data seen by the personnel division and the pay-roll division is partly identical This is expressed by the ***identity constraint***

Employees[#Emp,Name] id Salaries[#Id,Name]'

In the personnel division, information about all employees must be available, whereas in the marketing division this information is needed only for exactly those employees, working in that particular division This is specified by the ***selection constraint***

Marketing-emps[#Emp,Name,Address] sel
(Div='marketing' Employees)[#Emp,Name,Address]'

We assume that every employee is attached to a unique division Thus, the data belonging to the Marketing-emps scheme and the data related to the Stock-emps scheme are disjoint, although both schemes describe the same type of information on employees This is expressed by the ***disjoint constraint***

Marketing-emps[#Emp] disj Stock-emps[#Emp]'

Assume, most articles that are sold (and only such articles) are in stock, but sometimes there may be articles out of stock This is specified by the ***containment constraint***

Stock[#Prod,Name] con Articles[#Article,Name]'

To summarize all connections and relationships that belong to our example, we present the following set I of integration constraints

Employees[#Emp,Name] id Salaries[#Id,Name],
Pay-roll-emps[#Id,Address] sel
(Div='pay-roll' Employees)[#Emp,Address],

Marketing-emps[#Emp Name,Address] sel
(Div='marketing' Employees)[#Emp,Name Address]
Stock-emps[#Emp,Name Address] sel
(Div='stock' Employees)[#Emp,Name Address],
Pay-roll-emps[#Id] disj Marketing-emps[#Emp],
Pay-roll-emps[#Id] disj Stock-emps[#Emp],
Marketing-emps[#Emp] disj Stock-emps[#Emp],
Stock[#Prod,Name] con Articles[#Article,Name] •

Now, we present the general ***syntax*** of the integration constraints

Let V1, ,Vn be a set of proper database schemes, each representing a different view An ***integration constraint over V1, ,Vn*** is a statement of one of the following forms

- ***identity constraints*** (IDs) $R_i[Y]$ id $R_j[Z]$,
where R_i and R_j belong to different views and for some $Y' \subset P_i$, $Z' \subset P_j$ ($Y=K_iY'$, $Z=K_jZ'$, $dom(K_i)=dom(K_j)$ and $dom(Y')=dom(Z')$),

- ***selection constraints*** (SELs) $R_i[Y]$ sel (b R_j)[Z],
where R_i and R_j belong to different views, b is a condition for a selection on the attributes of R_j and for some $Y' \subset P_i$, $Z' \subset P_j$ ($Y=K_iY'$, $Z=K_jZ'$, $dom(K_i)=dom(K_j)$ and $dom(Y')=dom(Z')$),

- ***disjoint constraints*** (DISJs) $R_i[Y]$ disj $R_j[Z]$,
where R_i and R_j belong to different views, $Y=K_i$, $Z=K_j$ and $dom(K_i)=dom(K_j)$,

- ***containment constraints*** (CONs) $R_i[Y]$ con $R_j[Z]$,
where R_i and R_j belong to different views and one of the following conditions holds
a) for some $\phi \neq Y' \subset P_i$, $\phi \neq Z' \subset P_j$
($Y=K_iY'$, $Z=K_jZ'$, $dom(K_i)=dom(K_j)$ and $dom(Y')=dom(Z')$),
b) ($Y=K_i$ and ($Z \subset K_j$ or $Z \subset P_j$)) or ($Z=K_j$ and ($Y \subset K_i$ or $Y \subset P_i$)),

In order to give the ***semantics*** of these integration constraints we need the notion of an extended database scheme

An ***extended database scheme D*** is a triple D=<R|C|I>, where <R|C> is a proper database scheme which is extended by a finite set of integration constraints I
A database d over **R** is an ***instance of the extended scheme D***, iff $d \in sat(C) \cap sat(I)$, where
$sat(R_i[Y]$ id $R_j[Z]) = sat(R_i[Y] \subset R_j[Z]) \cap sat(R_j[Z] \subset R_i[Y])$,
$sat(R_i[Y]$ con $R_j[Z]) = sat(R_i[Y] \subset R_j[Z])$,
$sat(R_i[Y]$ disj $R_j[Z]) = sat(R_i[Y] \| R_j[Z])$,
$d=(r_1, ,r_m) \in sat(R_i[Y]$ sel (b $R_j)[Z])$
$\Leftrightarrow r_i[Y]=(\sigma_b\, r_j)[Z]$
$sat(I) = \bigcap_{i \in I} sat(i)$
Again, the ***set of all instances of D*** is denoted by I(D)

Remarks

- Sometimes we will regard a proper database scheme D=<R|C> as an extended database scheme D=<R|C|I> with $I=\phi$, and vice versa
- We note that we do not assume any implicit connections and relationships through names of attributes or relations Although equal names surely indicate possible connections, they should ***explicitly*** be specified
- Besides the possibilities to specify connections and relationships between the views through integration constraints, there is the additional possibility to add several manager views' to the set of all views These manager views can then be connected with the other normal views through additional integration constraints In our approach, we will not distinguish the different types of views

Note, that for some integration constraints, namely DISJs and CONs of the form b), there exist corresponding integrity constraints which conform to the restrictions, required for proper database schemes These are called ***simple integration constraints***, whereas the others are called ***pure integration constraints*** So, pure integration constraints are more powerful than the integrity constraints that are used in proper database schemes They are needed to describe situations in which even properties of objects are redundantly seen in different views, whereas these situations should be avoided within a single, proper database scheme

2 4 CONFLICTFREE VIEWS

Now, we are able to give a precise definition of what we understand by the conflictfree combination of several views to describe all valid global situations

Let V1=<**R1|C1**>, ,Vn=<**Rn|Cn**> be a set of proper database schemes with pairwise disjoint sets of relation schemes Furthermore, let **I** be a finite set of integration constraints over V1, ,Vn
The ***combination of V1, ,Vn with respect to I*** is the extended database scheme Comb(V1, ,Vn,**I**) =<**R|C|I**>, where $\mathbf{R} = \bigcup_{i=1\ n} \mathbf{R_i}$ and $\mathbf{C} = \bigcup_{i=1\ n} \mathbf{C_i}$

We often write 'Comb' instead of Comb(V1 Vn **I**)' whenever V1 ,Vn and **I** are understood Instances of Comb are also called ***valid global situations*** Thus, a valid global situation is a combination of valid view-instances which additionally satisfies all integration constraints

Naturally, a user of a specific view expects that every instance of his own view is represented by at least one valid global situation If there exists an instance of a view that cannot be combined with others to form a valid global situation, then we argue for assuming at least one of the views is not correctly specified For it allows a view database that is meaningless with respect to the whole application Or, much more harmfully, there exist conflicts between the views However, both cases should be avoided This motivates the following definitions

Let Comb=<**R|C|I**> be the combination of V1, ,Vn with respect to **I** Furthermore let d=(r11, ,r1l_1, , , rn1 rnl_n) be an instance of Comb, where r11, r1l_i are associated to the relation schemes of V_i
For $1 \leq i \leq n$, the ***projection of d to Vi*** is defined by
$d[V_i] = (r_{i1}, ,r_{il_i})$
Similiarly, the ***projection of I(Comb) to Vi*** is defined by
$I(Comb)[V_i] = \{d[V_i] \mid d \in I(Comb)\}$
The ***views V1, ,Vn are conflictfree with respect to I***, iff for all $i \in \{1, ,n\}$ $I(Comb)[V_i]=I(V_i)$

In the known literature, conflicts between the views are often informally described and simply reduced to be naming conflicts We note that our formal notion of conflictfreeness only depends on structural connections between the views, while different user groups are still allowed to choose their own appropriate names

Example (continued) It is easily shown that the views of our example are conflictfree with respect to the given set **I** of integration constraints Any instance of any view can obviously be extended according to **I**, in order to form a valid global situation, so that $I(Comb)[V_i]=I(V_i)$ is satisfied for all views V_i •

2.5 INCLUSION AND EQUIVALENCE OF DATABASE SCHEMES

Next, we motivate our notion of inclusion and equivalence of database schemes which we believe to be suitable in the field of view integration. In the literature, one finds several approaches to this problem, for instance [AABM], [Conn], [Hull], [Riss]. We use a slightly modified version of 'weakly included' as defined by Atzeni/Ausiello/Batini/Moscarini [AABM].

Intuitively, a database scheme G supports another database scheme V, if there exists a mapping f which associates every instance $g \in I(G)$ with an instance $f(g) \in I(V)$. Additionally, $f(I(G))=I(V)$ is required, so every instance of V is supported by at least one instance of G. In the field of view integration only instances of a global scheme G are physically stored, while the corresponding view-instances should easily be computed via f. Thus, we further require that f is given by an expression of the relational algebra. More precisely, f is an n-tupel of queries $f=(f1, ..., fn)$, where $f(g)=(f1(g), ..., fn(g))$. This leads to the following definitions:

Let $V=\langle\{R1[X1], ..., Rn[Xn]\}|C\rangle$ and $G=\langle\{R1'[X1'], ..., Rm'[Xm']\}|C'\rangle$ be two database schemes.
G supports V (or ***V is included in G***), denoted by $V \leq G$, iff there exists a relational query $f=(f1, ..., fn)$ $f(I(G))=I(V)$.
G and V are equivalent, denoted by $V \equiv G$ iff $V \leq G$ and $G \leq V$.
These definitions are obviously extended to include extended database schemes.

Now suppose, a global scheme G supports a view V via f and only the instance g of G is physically stored. A query q against V can be evaluated by first using f to compute the corresponding view instance $v=f(g)$ and thereafter, the desired answer q(v) is computed. But much more efficiently, we will first take the complete query q of and optimize it to get an equivalent query q' against G which is then evaluated to the answer $q'(g)=q(v)$.

2.6 THE FORMALIZED VIEW INTEGRATION PROBLEM

Now, we motivate our formal description of the view integration problem. Let V1, ..., Vn be the set of views that have to be integrated and let I be the corresponding set of integration constraints, specifying the connections between the views. A global database scheme G is sought which describes all valid global situations and simultaneously supports all views.

Obviously, Comb(V1, ..., Vn, I) is such a database scheme, but it is rather useless as a global scheme because of several deficiencies. Comb includes many complicated integration constraints. On the one hand, these constraints cause avoidable redundancy in the instances of Comb which makes them rather inefficient to be stored. On the other hand, in case of updates, additionally all integration constraints have to be checked which is rather difficult and inefficient, too.

To avoid these difficulties, we demand G to be a proper database scheme without integration constraints, simultaneously supporting all views, i.e., $Comb \leq G$. This requirement implies the existence of a query f against G such that $f(I(G))=I(Comb)$. Moreover f is also ***explicitly*** needed for the transformation of queries against the views, as described above.

Finally, we believe $G \leq Comb$ to be another appropriate property of the desired global scheme G, because instances of G should only represent such information, that is really used in valid global situations and nothing else. So, this requirement assures a kind of minimality of G. To sum up, we have the following definition:

Let V1, ..., Vn be a set of proper database schemes and let I be a set of corresponding integration constraints.
A ***solution to the view integration problem*** (given by V1, ..., Vn and I) is a pair (G,f), where
a) G is a proper database scheme,
b) Comb=G,
c) f is a relational query such that $f(I(G))=I(Comb)$.

We finish this section with a simple result, showing that in case the views are conflictfree, a solution to the view integration problem can really be used to support all views.

Proposition 2.1 Let V1, ..., Vn be a set of conflictfree, proper views with respect to a set I of integration constraints. Furthermore, let (G,f) be a solution to the given view integration problem.
Then G supports every view, i.e., $Vi \leq G$, for i=1, ..., n.

Proof straightforward by showing $Vi \leq Comb$ in case the views are conflictfree •

3 THE INTEGRATION METHOD

3.1 INTRODUCTION

In this paragraph, we develop our method to solve view integration problems which is based on a more general philosophy for database design [Bisk].

The required properties of a solution are split into a 'basic' property Γ that is easy to achieve and an 'additional' property Ω which is much more difficult to achieve. In a first step, an initial design satifying the basic property is produced. In a second step, this initial design is improved step-by-step by local transformations which eliminate 'forbidden substructures' until the additional property is satisfied, while the basic property is still preserved.

In our particular case, we have a set V1, ..., Vn of proper database schemes and a set I of corresponding integration constraints. A solution to the given view integration problem is a pair (G,f), and we use

- as basic property $\Gamma((G,f))$ <==>
  G=<R|C|I> is an extended database scheme such that G=Comb and
  f is a relational query against G such that $f(I(G))=I(Comb)$,
- as additional property $\Omega((G,f))$ <==>
  G is a proper database scheme,
- as forbidden substructure
  an integration constraint $i \in I$

In this particular case, the first step of the above design philosophy is rather trivial. Obviously, (Comb,id) already satisfies the basic property, where 'id' denotes the identity query against Comb. So, we will use (Comb,id) as the initial design which has to be improved by the second step. The basic notions, needed for the formal definition of the desired local transformations, are shortly described by using our former example.

Example (continued) The following relation schemes and the corresponding integration constraint form a local part of the extended database scheme Comb.
Employees[#Emp,Name,Address,Date,Div]
Salaries[#Id,Name Sal,#Acc] and
Employees[#Emp Name] id Salaries[#Id Name]

This identity constraint, forming a forbidden substructure, can simply be eliminated by merging both relation schemes together to get a new one representing both old schemes

Emp-Sal[#Emp,Name Address,Date,Div Sal #Acc]

This transformation not only eliminates the undesired identity constraint, but simultaneously avoids the redundancy induced by it ▪

There are similiar transformations for all other kinds of integration constraints For instance, a selection constraint $R_i[K_iP_i]$ sel (b R_j)$[K_jZ]$' will be eliminated by removing the redundant relation scheme R_i A containment constraint $R_i[K_iY]$ con $R_j[K_jZ]$' will be simplified by removing the redundant attributes Y in R_i Finally, a simple integration constraint is eliminated by replacing it by its equivalent integrity constraint

But we are still faced with another problem, because whenever a relation scheme is changed by such a transformation, we have to guarantee that all other constraints, belonging to this scheme, can be transformed correspondingly That is, in general the relevant local part of an extended database scheme, changed by a transformation, is somewhat larger than it is indicated by the above example

So, in general a transformation is divided into the following steps

- choose an integration constraint that should be eliminated,
- compute the set of all constraints (i e , integrity and integration constraints), forming the relevant local part for the intended transformation,
- check, whether the above set of constraints can be transformed correspondingly,
- finally, perform the complete transformation

Most of those constraints that form the relevant local part of an intended transformation can easily be transformed correspondingly However, sometimes there are some so-called disturbing' constraints, that cannot be changed Having the above proposed transformations in mind, we can give a simple syntactical characterization of those disturbing constraints

Let **B** be a set of constraints (i e , integrity or integration constraints), and let i be an integration constraint

disturbing(**B** i) =

$$\begin{cases} \{b\in\mathbf{B} \mid b\neq i,\ b \text{ refers to both } R_j \text{ and } R_l\}, \\ \qquad \text{if } i\equiv R_j[K_jY] \text{ id } R_l[K_lZ], \\ \{b\in\mathbf{B} \mid b\neq i,\ b \text{ refers to } R_j\}, \\ \qquad \text{if } i\equiv R_j[K_jP_j] \text{ sel } (b' R_l)[K_lZ], \\ \{b\in\mathbf{B} \mid b\neq i,\ b \text{ refers to both } R_j \text{ and to an attribut of } Y\}, \\ \qquad \text{if } i\equiv R_j[K_jY] \text{ con } R_l[K_lZ] \text{ and } Y,\ Z\neq\phi \\ \phi \qquad \text{otherwise} \end{cases}$$

not_disturbing(**B**,i) =**B**\disturbing(**B**,i)

In case there exist disturbing constraints, an intended transformation is performable only if those constraints are redundant

More formally, let D=<**R**|**C**|**I**> be an extended database scheme and let i be an integration constraint, according to an intended transformation

performable(D i) <==>
i∈**I** and not_disturbing(**C**∪**I**,i) |= disturbing(**C**∪**I**,i),
where |=' denotes ***finite*** logical implication [FaVa]

3 2 THE SCHEME TRANSFORMATIONS

Now, we will give a precise definition of the local transformations that are used to solve view integration problems

Let **EDBS** be the set of all extended database schemes which are denoted by D=<**R**|**C**|**I**> or D'=<**R'**|**C'**|**I'**>

i) $\text{----->}_{id} \subset \mathbf{EDBS}\times\mathbf{EDBS}$, where D ----->_{id} D' <==>

there exists an integration constraint
$i\equiv R_j[K_jY]$ id $R_l[K_lZ]$, such that
$K_jP_j\cap(P_l\setminus Z)=\phi$,
performable(D,i),
R'=**R**\{R_j,R_l} ∪ {$R_j[K_jP_j(P_l\setminus Z)]$},
C'={b'|there exists an b∈not_disturbing(**C**,i) and
b' results from b after the renaming according
to ($R_l[K_lZ]$ ==> $R_j[K_jY]$)},
I'={b |there exists an b∈not_disturbing(**I**,i)\{i} and
b' results from b after the renaming according
to ($R_l[K_lZ]$ ==> $R_j[K_jY]$)},

ii) $\text{----->}_{sel} \subset \mathbf{EDBS}\times\mathbf{EDBS}$, where D ----->_{sel} D' <==>

there exists an integration constraint
$i\equiv R_j[K_jP_j]$ sel (b R_l)$[K_lZ]$, such that
performable(D,i),
R'=**R**\{R_j},
C'=not_disturbing(**C**,i),
I'=not_disturbing(**I**,i)\{i},

iii) $\text{----->}_{con} \subset \mathbf{EDBS}\times\mathbf{EDBS}$, where D ----->_{con} D' <==>

there exists an integration constraint
$i\equiv R_j[K_jY]$ con $R_l[K_lZ]$, such that
Y, Z≠ϕ,
performable(D,i),
R'=**R**\{R_j} ∪ {$R_j[K_j(P_j\setminus Y)]$},
C'=not_disturbing(**C**,i),
I'=not_disturbing(**I**,i)\{i} ∪ {$R_j[K_j]$ con $R_l[K_l]$},

iv) $\text{----->}_{sim} \subset \mathbf{EDBS}\times\mathbf{EDBS}$, where D ----->_{sim} D' <==>

there exists a simple integration constraint i, such that
performable(D,i),
R'=**R**
C'=**C**∪{c_i}, where c_i is the integrity constraint
equivalent to i,
I'=**I**\{i},

v) --> ⊂ **EDBS**×**EDBS**, where

--> = ----->_{id} ∪ ----->_{sel} ∪ ----->_{con} ∪ ----->_{sim}

Remarks

- For simplicity of notation, we choose the above naming conventions in the ----->_{id}-transformations Sometimes other names may be more suitable for the new relation scheme, but of course, later renamings are still possible
- Note, that a selection constraint can only be eliminated, if a complete relation scheme stands on the left hand side of the selection constraint Hence often, one first has to remove some attributes by transformations of ----->_{con} before a ----->_{sel}-transformation can be used
- Kent also studies scheme transformations [Kent] but with another aim in mind His transformations should help covering a wide range of possible different design options whereas the above transformations can only be used to simplify database schemes by eliminating complicated integration constraints

Now, we are able to show that the above scheme transformations have all those properties that are demanded by our proposed design approach

Theorem 3 1 Let D and D′ be two extended database schemes such that D-->D′
Then D≡D′ and a query f against D′ with f(I(D′))=I(D) can effectively be computed from the eliminated integration constraint

Proof see Appendix ▪

This important result shows that the transformations can be used to simplify the initial design (Comb,id) until the additional property Ω is achieved, while the basic property Γ is preserved

Corollary 3 2 Let V1, ,Vn be a set of proper database schemes and let **I** be a set of corresponding integration constraints Furthermore, let Comb=Go--> -->Gn=G be a sequence of scheme transformations, where G is a proper database scheme For i=1, ,n let fi be a query against Gi, where f(I(Gi))=I(G(i-1)) according to the last theorem
Then (G,f) is a solution to the given view integration problem, where f =f1o ofn

Proof by induction and the above theorem ▪

In determining a sequence of scheme transformations to simplify an extended database scheme to a proper one, ----->_{sim}-transformations can be ignored until only simple integration constraints are left This is a special case of the next theorem

Theorem 3 3 For i=0, ,n, n>0, let Di be some extended database schemes such that Do-->D1--> -->Dn
Then there exist l, m∈$\mathbf{N}_0$ and some extended database schemes Di′, where
0≤l≤m≤n,
Do′-->D1′--> -->Dm′,
Do′=Do, Dm′=Dn,
(Di′,D(i+1)′) ∈ $\text{-->\-----}>_{sim}$ for 0≤i<l,
(Di′,D(i+1)′) ∈ ----->_{sim} for l≤i<m

Proof see the full paper [BiCo] ▪

3 3 THE INTEGRATION METHOD

Now, we sketch our integration method, as developed above

In the first step, (Comb,id) is taken to be the initial design satisfying the basic property Γ In the second step, we try to achieve the additional property Ω by eliminating all integration constraints with our local scheme transformations According to the last theorem, we first try to eliminate all pure integration constraints and thereafter, the remaining simple ones are transformed

In this second step, there is a kind of nondeterministic choice, when selecting any performable transformation We propose, that this choice should be guided interactively by the designer Often, it is useful to select those transformations, which result in the most significant simplifications I e , ----->_{sel}-transformations should have the highest priority, followed by ----->_{id}-transformations, which are followed by transformations belonging to ----->_{con}

Whenever there are still pure integration constraints left, although no corresponding transformation is performable, an interactive exception handling must be started to solve the situation manually or to stop the integration for a more detailed investigation of the problem

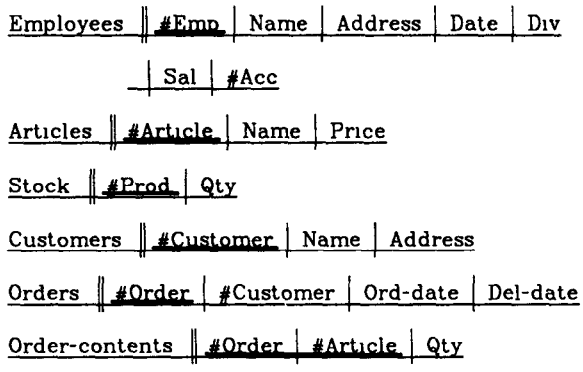
The whole method is presented again as a frame of a program

```
PROGRAM integrate,
INPUT (*a view integration problem*)
  V1, ,Vn  proper database schemes,
  I  set of integration constraints,
OUTPUT (*a solution to the view integration problem*)
  G  proper database scheme,
  f  query against G,
VAR
  stop  BOOLEAN,
BEGIN
  stop  = FALSE,
  READ(V1, ,Vn,I),
  G =Comb(V1, ,Vn,I),
  f =id,
  WHILE there are still pure integration constraints left
        AND (NOT stop) DO
  BEGIN
    SELECT ANY pure integration constraint i
                such that performable(G,i),
    IF any such pure integration constraint could be
       selected
      THEN
        perform the corresponding transformation on G
        and update f accordingly
      ELSE
        exection_handling(stop),
        (*stop is possibly changed to TRUE *)
  END, (*WHILE*)
  IF stop
    THEN
      WRITE('Stopped during exception handling ')
    ELSE
    BEGIN
      transform the remainding simple integration
      constraints,
      WRITE(G,f),
    END,
END
```

The View Integration Algorithm

Example (continued) At first, all selection constraints can be eliminated by the corresponding ----->_{sel}-transformations because the disturbing disjoint constraints are all redundant Thereafter, an ----->_{id}-transformation can be used to eliminate the identity constraint Similarly, the containment constraint is simplified by the associated ----->_{con}-transformation At least, the remaining simple integration constraint is transformed to its equivalent integrity constraint and we get the following solution (G,f)

G



Stock[#Prod]⊂Articles[#Article],
Orders[#Order]⊂Order-contents[#Order],
Orders[#Customer]⊂Customers[#Customer],
Order-contents[#Order]⊂Orders[#Order],
Order-contents[#Article]⊂Articles[#Article]

f is chosen according to the following assignments

(The relation names on the left side belong to the relation schemes of Comb, whereas the names on the right side belong to the relation schemes of G)

V1 Employees <--
Employees[#Emp,Name,Address Date,Div]

V2 Salaries <-- Employees[#Emp,Name,Sal,#Acc]
Pay-roll-emps <--
$(\sigma_{\text{Div= pay roll}}$ Employees)[#Emp,Address]

V3 Articles <-- Articles,
Customers <-- Customers,
Orders <-- Orders
Order-contents <-- Order-contens,
Marketing-emps <--
$(\sigma_{\text{Div= marketing}}$ Employees)[#Emp,Name,Address]

V4 Stock <--
(Stock $\bowtie_{\#\text{Prod}=\#\text{Article}}$ Articles)[#Prod Name,Qty],
Stock-emps <--
$(\sigma_{\text{Div= stock}}$ Employees)[#Emp,Name,Address]

Now, G can be used as a global database scheme, supporting the views of all different divisions and with the help of f queries against any of these views can easily be translated into queries against G •

4 CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

The main goal of our work has been to develop a computer-aided view integration tool, based on a strictly formal background Therefore in Section 2, we have chosen a data model with a clear formal semantics, to specify both the user views and the global database scheme Then we have introduced integration constraints as a formal means for specifying the connections and relationships between the views These integration constraints are strictly distinguished from simple integrity constraints, because of a different intended meaning Then we have given a formalized notion of conflictfreeness of a set of views with respect to a specification of their connections and relationships After introducing a formalized concept of inclusion and equivalence of database schemes, we have presented a complete formal description of the view integration problem Particularly, this clarifies the details 1) - 5), as mentioned in Section 1

On this formal basis, we have developed our integration approach, which starts with the combination of all views and the specified connections and relationships This initial design is simplified step-by-step by local scheme transformations until a solution to the given integration problem is found In Section 3, we have presented those local, equivalence preserving scheme transformations and proved their basic properties Finally, we have sketched our algorithm for a computer-aided view integration

There are still various problems left open, which should be investigated in some further research

Decision of Conflictfreeness When using our approach in real world applications, one surely wishes for an automated decision algorithm for the conflictfreeness of the specified views with respect to the corresponding set of integration constraints Unfortunately, conflictfreeness in general is undecidable [Conv2], so that we have to look for a decidable subclass which is still of great practical relevance In all other cases conflictfreeness must still be proved manually We note that the decision of conflictfreeness is closely related to the following problem

Implication of Constraints When checking whether a transformation according to a given integration constraint i is applicable, we have to decide whether performable(D,i), i e , to decide the implication problem not_disturbing(C∪I) |= disturbing(C∪I)' (see 3 1) These implication problems are known to be rather difficult [CaVi], [CFP], [ChVa], [FaVa], [Mitc] Indeed, in [Conv2] finite logical implication for the used class of integrity and integration constraints is shown to be undecidable in general So, for practical reasons, we propose to use an efficiently implementable, incomplete version weakly_performable(D,i) with 'weakly_performable(D,i) ==> performable(D,i)', which should be able to show the logical implications at least for simple, often occuring special cases, using suitable heuristics Preliminary results to these implication problems can be found in [Conv1]

Updates through Views In our approach, we solely considered static aspects of database design For concrete real world applications, dynamic aspects, especially updates through views, have to be investigated, too This problem alone is known to be rather complicated and is still a topic of current research [CoPa], [FuCa], [Kell], [Stie]

Currently, we are involved in implementing a prototype design tool for both view modelling and view integration

APPENDIX

Proof of Theorem 3 1

Let D=<R|C|I> and D'=<R'|C'|I'> be two extended database schemes such that D-->D' According to the eliminated integration constraint, there are several cases

Case 1 D-----$>_{id}$D' according to i≡Rj[KjY] id Rl[KlZ]

W l o g , we assume that R, R', j and l are of the following forms

R={R1[K1P1], ,R(n-1)[K(n-1)P(n-1)],Rn[KnPn]},
R'={R1'[K1'P1'], ,R(n-1)'[K(n-1)'P(n-1)']},
j=n-1 and l=n

Then we have

Rk'=Rk, for k=1, ,n-2,
Kj'=Kj and
Pj'=Pj(Pl\Z)

Now, define queries f and f' as follows

f Rk <-- Rk', for k=1, ,n-2,
Rj <-- Rj'[KjPj] and
Rl <-- Rj [KjY(Pl\Z)]

f' Rk' <-- Rk, for k=1, ,n-2,
Rj' <-- (Rj $\bowtie_{KjY=KlZ}$ Rl)[KjPj(Pl\Z)]

At first, we prove a) f'(I(D))=I(D')

⊂' For d=(r1, ,r(n-2),rj,rl)∈I(D) and d =(r1', ,r(n-2)',rj') =f'(d) we have to show that d'∈I(D')

Since d satisfies i rj and rl are consistent, i e , they join completely So, we have
rj [KjPj]⊃rj and
rj'[KjY(Pl\Z)]⊃rl
Moreover d satisfies the FDs Rj Kj->Pj and Rl Kl->Pl and rj' is the result of the equijoin of rj and rl on the superkeys KjY and KlZ Thus, d' satisfies the FD Rj' Kj->Pj(Pl\Z) and we have
rj'[KjPj]=rj and
rj'[KjY(Pl\Z)]=rl

((*) Note this implies f(f'(d))=d, for all d∈I(D)')

Since rj' contains rj and rl as projections, and since d satisfies **C** and **I**, it follows that d' satisfies the inherited' constraints **C'** and **I'**, i e d'∈I(D')

⊃' For d'=(r1' ,r(n-2)' rj')∈I(D) we define d=(r1, ,r(n-2),rj rl) =f(d') We have to show that d∈I(D) and f'(d)=d'

((**) Note, this implies f'(f(d'))=d', for all d'∈I(D')')

Since d' satisfies the inherited' constraints **C'** and **I'**, and since rj and rl are suitable projections of rj', d satisfies not_disturbing(C∪I,i)\{i} Moreover, i is obviously satisfied by d, and hence all constraints in C∪I, since not_disturbing(C∪I,i) |= disturbing(C∪I,i) I e , d∈I(D)
Since both projections rj and rl include the key attributes, rj' is losslessly decomposed into rj and rl, i e , rj =(rj $\bowtie_{KjY=KlZ}$ rl)[KjPj(Pl\Z)] Thus f (d)=d'

Now, we prove b) f(I(D'))=I(D)
From a) it follows that f(f (I(D)))=f(I(D')) and using (*) we get I(D)=f(I(D'))

To summarize a) and b), we have D≡D' which completes Case 1

Case 2 D-----$>_{sel}$D' according to i≡Rj[KjPj] sel (b Rl)[KlZ]

Define f and f' as follows

f Rk <-- Rk' for k≠j,
Rj <-- (σ_b Rl')[KlZ]

f' Rk <-- Rk, for k≠j

Again, it is easily shown that the queries have the required properties

Case 3 D-----$>_{con}$D' according to i≡Rj[KjY] con Rl[KlZ]

Define f and f' as follows

f Rk <-- Rk', for k≠j,
Rj <-- (Rj' $\bowtie_{Kj=Kl}$ Rl')[KjZ(Pj\Y)]

f' Rk' <-- Rk, for k≠j,
Rj' <-- Rj[Kj(Pj\Y)]

Again, it is easily shown that f and f' have the required properties

Case 4 D-----$>_{sim}$D' according to a simple integration constraint

In this case D and D' are obviously equivalent, where f and f' are the identity queries ▪

Remarks

- In all cases, f and f' are inverses of one another Hence, D and D' are equivalent even according to some stronger notions of equivalence, as for instance presented in [AABM] and [Hull]
- Only a subset of the relational operations are used to define the queries f and f' f is defined using projections, equijoins and selections and f' is defined using only projections and equijoins This will be important when considering the view update problem!

REFERENCES

[AABM] P Atzeni, G Ausiello, C Batini, M Moscarini, Inclusion and Equivalence between Relational Database Schemata', Theoretical Computer Science 19, 1982 267-285

[BaLe] C Batini, M Lenzerini, A Methodology for Data Schema Integration in the Entity-Relationship Model', in Entity-Relationship Approach to Software Engineering', C G Davis, S Jajodia, P A Ng, R T Yeh (eds), North-Holland, 1983, 413-420

[BDB] J Biskup, U Dayal, P A Bernstein, Synthesizing Independent Database Schemas', ACM-Sigmod International Conference on Management of Data, 1979, 143-151

[BiCo] J Biskup, B Convent, A Formal View Integration Method', Forschungsbericht Nr 208, Universitat Dortmund, 1985

[Bisk] J Biskup, Entwurf von Datenbankschemas durch schrittweises Umwandeln verbotener Teilstrukturen', GI-EMISA-Fachgesprâch Entwurf von Informationssystemen - Methoden und Modelle', 1985, 130-148

[BLM] C Batini, M Lenzerini, M Moscarini, Views Integration', in Methodology and Tools for Data Base Design', S Ceri (ed), North-Holland, 1983, 57-84

[Brod] M L Brodie, On the Development of Data Models', in 'On Conceptual Modelling', M L Brodie, J Mylopoulos, J W Schmidt (eds) Springer-Verlag, 1984, 19-48

[CaVi] M.A. Casanova, V.M.P. Vidal, 'Towards a sound View Integration Methodology', 2nd ACM-Sigact-Sigmod Symposium on Principles of Database Systems, 1983, 36-47.

[CFP] M.A. Casanova, R. Fagin, C.H. Papadimitriou, 'Inclusion Dependencies and their Interaction with Functional Dependencies', 1st ACM-Sigact-Sigmod Symposium on Principles of Database Systems, 1982, 171-176.

[ChVa] A.K. Chandra, M.Y. Vardi, 'The Implication Problem For Functional And Inclusion Dependencies Is Undecidable', SIAM J. Comput., Vol. 14, No. 3, 671-677, 1985.

[Codd1] E.F. Codd, 'A relational Model of Data for large shared Data Banks', Comm. ACM 13 (6), 1970, 377-387.

[Codd2] E.F. Codd, 'Further Normalization of the Data Base Relational Model', in 'Data Base Systems', R. Rustin (ed.), Courant Computer Science Symposia 6, Englewood Cliffs, N.J., Prentice-Hall, 1972, 33-64.

[Conn] T.S. Connors, 'Equivalence Of Views By Query Capacity', 4th ACM-Sigact-Sigmod Symposium on Principles of Database Systems, 1985, 143-148.

[Conv1] B. Convent, 'Ein formaler Ansatz zum rechnergestützten Entwurf von Datenbankschemata mittels View-Integration', Diplomarbeit, Universität Dortmund, 1984.

[Conv2] B. Convent, 'Unsolvable Problems Related To The View Integration Approach', in preparation.

[CoPa] S.S. Cosmadakis, C.H. Papadimitriou, 'Updates of Relational Views', 2nd ACM-Sigact-Sigmod Symposium on Principles of Database Systems, 1983, 317-331.

[Fagi] R. Fagin, 'Multivalued Dependencies and a New Normal Form for Relational Databases', ACM TODS 2 (3), 1977, 262-278.

[FaVa] R. Fagin, M.Y. Vardi, 'The Theory of Data Dependencies - A Survey', IBM Research Report RJ 4321, IBM Research Laboratory, San Jose, 1984.

[FuCa] A.L. Furtado, M.A. Casanova, 'Updating Relational Views', Technical Report 020, IBM do Brasil, 1984.

[Hull] R. Hull, 'Relative Information Capacity of Simple Relational Database Schemata', 3rd ACM-Sigact-Sigmod Symposium on Principles of Database Systems, 1984, 97-109.

[Kell] A.M. Keller, 'Updating Relational Databases through Views', Dissertation, Stanford University, 1985.

[Kent] W. Kent, 'Choices in practical Data Design', 8th VLDB, 1982, 165-180.

[Lum] V.Y. Lum et. al., '1978 New Orleans Data Base Design Workshop Report', 5th VLDB, 1979, 328-339.

[Maie] D. Maier, 'The Theory of Relational Databases', Computer Science Press, 1983.

[MaWi] R. El-Masri, G. Wiederhold, 'Data Model Integration using the Structural Model', ACM-Sigmod International Conference on Management of Data, 1979, 191-202.

[Mitc] J.C. Mitchell, 'The Implication Problem for Functional and Inclusion Dependencies', Information and Control 56, 1983, 154-173.

[NaGa] S.B. Navathe, S.G. Gadgil, 'A Methodology for View Integration in Logical Database Design', 8th VLDB, 1982, 142-164.

[Riss] J. Rissanen, 'On Equivalence of Database Schemes', 1st ACM-Sigact-Sigmod Symposium on Principles of Database Systems, 1982, 23-26.

[Stie] H. Stiefeling, 'Änderungen auf Sichten relationaler Datenbanken', Diplomarbeit, Universität Dortmund, 1984.

[TeFr] T.J. Teorey, J.P. Fry, 'Design of Database Structures', Prentice-Hall, 1982.

[Ullm] J.D. Ullman, 'Principles of Database Systems', Pitman, 1982.

[WiMa] G. Wiederhold, R. El-Masri, 'A Structural Model for Database Systems', Technical Report Stan-CS-79-722, Stanford University, 1979.

[YNW] S.B. Yao, S.B. Navathe, J.-L. Weldon, 'An Integrated Approach to Database Design', Data Base Design Techniques I, Requirements and logical Structures, NYU Symposium New York, 1978, 1-30.

[YWH] S.B. Yao, V.E. Waddle, B.C. Housel, 'View Modeling and Integration using the Functional Data Model', IEEE Trans. on Software Engineering, Vol. SE-8, No. 6, 1982, 544-553.