Quantifying Schema Evolution*

Dag Sjøberg, Department of Computing Science, University of Glasgow,
17 Lilybank Gardens, Glasgow G12 8QQ, Scotland, sjoberg@dcs.glasgow.ac.uk

Abstract

Achieving correct changes is the dominant activity in the application software industry. Modification of database schemata is one kind of change which may have severe consequences for database applications. The paper presents a method for measuring modifications to database schemata and their consequences, by using a thesaurus tool. Measurements of the evolution of a large-scale database application currently running in several hospitals in the UK are presented and interpreted. The kind of measurements provided by this in-depth study is useful input to the design of change management tools.

Keywords: Schema evolution, change statistics, change management tools.

1 Introduction

One of the most challenging problems of building and maintaining large, long-lived application systems is to cope with all the changes that inevitably will be imposed on the systems over time. Many large application systems are centred around a database. The description of the data that can be represented in a database is called the "database schema". A particular kind of change that may have serious consequences for the rest of the application systems is change to such schemata.

In order to acquire a deeper understanding of the nature of schema evolution, *measurements* of change in an actual system under development and in operational use were collected. A measuring tool, the *thesaurus*, was built to monitor the evolution of a large, industrial database application – a health management system (HMS). This system is centred around a relational database and has user interfaces built on top of the X Window System. The HMS system was observed over a period of 18 months. The paper reports how the schema changed and furthermore shows that even a small change to the schema may have major consequences for the rest of the application code. The measurements confirm the need for tools and techniques for managing the consequences of changes to database schemata. Moreover, the measurements identify more precisely the requirements of such tools and techniques.

The purpose of this paper is to present a research direction concerning the problem of *quantifying* schema evolution. The study reported has wider applications than just to traditional

* Published in: *Information and Software Technology*, Vol. 35, No. 1, pp. 35-44, January 1993

database systems. The data descriptions and consequently dependent data (including programs) of all persistent application systems will inevitably have to be changed in order to reflect the changing user needs. That is, schema evolution in traditional databases corresponds to class evolution in object-oriented database systems, to type evolution in applications developed in strongly-typed persistent programming languages (e.g. Napier88 [Morrison *et al.* 1989]) and, at a higher level, to changes to application models described in the framework of conceptual data models (e.g. the Entity-Relationship Model [Chen 1976]).

The remainder of this introduction contains a more detailed description of the concept of schema evolution and its impact on the rest of the application. Section 2 presents an overview of the HMS application and describes how the thesaurus tool has been designed to measure schema evolution and its consequences. The measurements are presented and interpreted in Section 3. Due to changing development environments it was necessary to change the thesaurus tool itself during the period of investigation; examples of this kind of change are discussed in Section 4. Section 5 summarises the study and sketches further work in this area.

1.1 Schema Evolution

It is well known that attempting to achieve correct changes is the dominant activity in the application software industry [Zelkowitz 1978, Putnam 1982, Parikh and Zvegintsov 1983, Corbi 1989, Chikofsky and Cross 1990]. Schema changes constitute one very important category of changes. There are several sources of such changes, for example:

i) People do not know in advance, or are not able to express, all the desired functionality of a large-scale application system. Only experience from using the system will enable the needs and requirements to be properly formulated.

ii) The application world is continually changing. A viable application system must be enhanced to accommodate these changes.

iii) Often the scale of the task requires incremental design, construction and commissioning. This results in requirements to change the installed subsystems.

Consequently, continuous modifications to the schemata are necessary to ensure that the system reflects the requirements as accurately as possible at all times.

Relational database management systems (RDBMS) are currently in widespread use in industry and commerce. The HMS system is one example of an application system utilising this technology. RDBMSs are based on the *relational data model* [Codd 1970], and a database built with a RDBMS is called a relational database and consists of relations (also called tables) and fields (also called attributes or columns). A list of logical changes[1] to a relational schema is:

1) Add a new relation.

[1] Physical re-organisation is not an issue in this paper as most RDBMSs absorb such changes obviating the need to change applications.

2) Rename a relation.
3) Delete a relation.
4) Add a new field to a relation.
5) Rename a field.
6) Change the type of a field.
7) Delete a field from a relation.

1.2 Consequences of Schema Evolution

Concerning change, the following principles should be pursued [Atkinson 1991]:

- Change should be accompanied by minimum consequential loss of information and minimal disruption of other components: Limit the propagation of unnecessary change.
- All consequences of change must be dealt with: Ensure the propagation of necessary change.

The way schema modifications are dealt with today is often *ad hoc*, and the necessary conversions may be expensive due to factors such as a requirement to shutdown the system, programmer effort, machine resources, etc.

The effects of schema changes are divided into three categories:

- effects on other parts of the schema,
- effects on extensional data, and
- effects on application programs.

Most literature on schema evolution [Banerjee *et al.* 1987, Penney and Stein 1987, Skarra and Zdonik 1987, Kim and Chou 1988, Joseph *et al.* 1989, Lerner and Habermann 1990] focuses on the first and second category, but these areas are not the subject of this paper. There is little reported research in the third category, and this is in stark contrast to its significance for application programmers. One of the intentions of this paper is thus to illustrate the extent of such change effects by presenting measurements of a large, real-world application system. Typically, there will be many application programs that utilise a type which has been changed in the schema. These programs may use screen definitions, query definitions, procedures, etc. It should not be difficult to imagine that incompatibilities between a schema type and the corresponding type assumed by the application programs may have serious consequences. For example when a field is added, at least one application program and screen must be changed to collect the new data, and at least one program must eventually use it.

Recent work is concentrated in the OODB area where the consequences of changing a type (class) may lead to more significant changes in the schema itself than in a relational environment, but the consequences for extensional data and application code may be as serious as in a relational environment.

2 The Thesaurus – A Tool for Measuring Schema Evolution

The measurements described in this paper were collected by the thesaurus tool [Sjøberg 1991]. This section presents an overview of the HMS system, for which the tool was built, and the basic features of the tool itself.

2.1 The HMS System

The HMS system, running on high resolution colour Unix™ workstations, consists of *Display Language* and *Hippo* programs [Clifton 1990, England and Selwyn 1990], a *query dictionary* and a database including the associated *schema* (Figure 2.1).

Applications are written as a graph of screens so that a user works via the icons and fields on screens and navigates to other screens in the graph using "buttons". The screens of the user interface are programmed in the Display Language. A Display Language program contains *classes* and *objects* that both represent windows and have attributes that describe properties of these windows. Objects can be defined within classes and within other objects. A class can be used as the type of another class or as the type of an object. It is possible to modify the type of an object by adding attributes or by introducing new objects within the original object in a form of inheritance hierarchy. The Display Language is an interpreted language implemented in C and the X Window System.

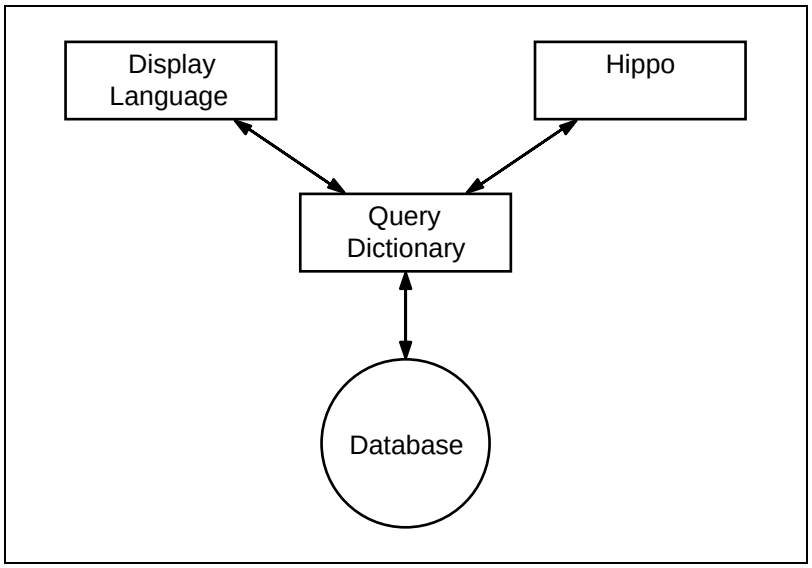

Figure 2.1: The main components of the HMS system

The procedural part of the user interface is programmed in the Hippo language. An *action* is the main language construct. An action can be global, or it can be local to a *script* which, in turn, may be associated with a main class in a Display Language program. Hippo is an interpreted language implemented in C.

The query dictionary consists of *queries* (SQL *select*) and *update functions* (SQL *insert, update* and *delete*) which are used by the Display Language and Hippo programs when operating on the database. Several update functions may be defined in a *transaction* (usually to ensure referential integrity after update). The query dictionary concept was introduced in the HMS architecture to isolate as far as possible the Display Language and Hippo code from the database. This permits some of the changes to the schema to be hidden from the application code by rewriting the queries and update functions. These queries and update functions are referred to by name with named parameters (Hippo variables) called *datums*[2]. The queries return their results in tables whose columns are also referred to as datums and which may be traversed or automatically displayed. The query dictionary is intended to be sufficiently general not only to absorb change that need not be propagated further, but also to allow different DBMSs to be used and even different data models. The query dictionary is implemented in the is implemented in the Pro*C™. embedded SQL language. The description of the relations, including their fields, constitutes the schema. The actual DBMS is Oracle™.

2.2 Cross-References

The thesaurus tool assists in keeping track of the use of names in the HMS application and helps answer questions such as: Which actions, classes, functions, macros, etc. are defined and where are they used? Which fields and relations does this query or update function refer to? Which actions are referenced in this Display Language program? etc. The information about the names is kept in the *Thesaurus* relation whose fields are described in Figure 2.2.

- NAME – a textual form of the entry
- SEQ_NO – system generated key
- NAME_TYPE – one of the following codes:
  Action Name (AN), Action Script name (AS), Class Name (CN), Datum Name (DN),
  Field Name (FN), FUnction name (FU), Query Name (QN), Relation Name (RN),
  Screen Macro name (SM), Transaction Name (TN) or Update function Name (UN)
- CONTAINER – a textual name describing where a name is used
- CONTAINER_TYPE – codes appropriate to the type of the CONTAINER value:
  Action Script (AS), Display Language program (DL), Hippo Program (HP), Query (QN),
  Query Dictionary (QD), Relation (RN), Schema (SC), Transaction (TN) or Update function (UN)
- DEFINITION_USE (D/U) – indicates definition or use of the name
- REMARK – a comment on the name

Figure 2.2: The Thesaurus *relation*

[2] Plural of *datum* is *data*, but HMS uses *datums* to denote several occurrences of the special HMS concept *datum*.

By November 1991, the HMS system comprised about 150,000 lines of source code, but the thesaurus provides a better measurement of the size: the number of programmer-introduced names of various types. Figure 2.3 shows the proportion of definitions and uses for each name type. In total there are 9152 defined names which are used 15098 times. These measurements describe the number of unique occurrences within a container type. That is, if for example a datum is referred to several times within an action, it is registered as only one entry in the thesaurus.[3] The apparently low use of action scripts and update functions should be explained. There are 168 action scripts that are called in the Hippo code. Another sort of use is that an action script may be associated with a class having the same name as the script. There are 128 such associations. Among the 322 defined update functions, 237 are contained in transactions and are thus only called implicitly when the associated transaction is called.

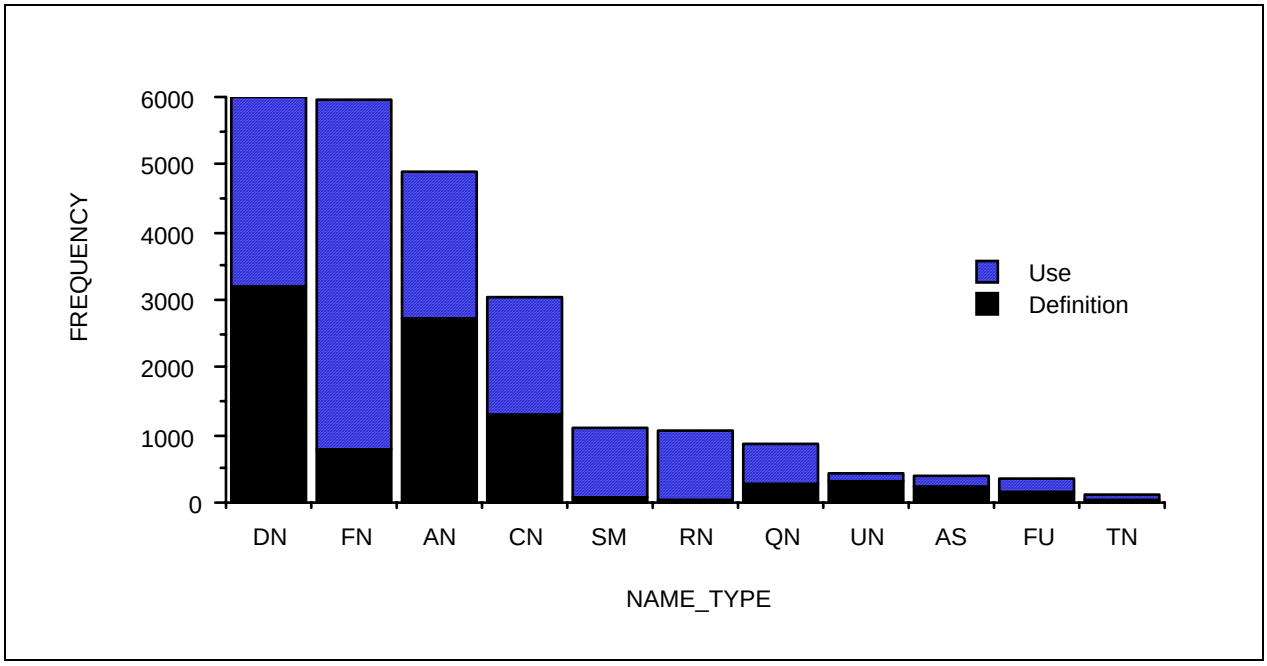

Figure 2.3: Definitions and uses of names distributed by NAME_TYPE

The part of the tool that generates the names and performs the subsequent updates of the thesaurus data has been implemented as a combination of UNIX *csh*, *awk* and *sed* scripts and one C program. An interface consisting of windows with pull-down menus, buttons, etc. is implemented in the Display Language and Hippo themselves. In addition to search and display of name information, the interface provides predefined queries for *consistency checks* like detecting names defined but not used, and worse, names used but not defined.

[3] Information about duplicated name occurrences within a container was not considered necessary for the HMS project. (Including duplicates would have increased the volume of the thesaurus by 100%.)

A definite requirement of the tool – which has been satisfied – was that the contents of the thesaurus should not need to be manually maintained. Experience shows that this is crucial for the use of such a tool. The source programs and database schema are periodically scanned (every night since December 1991) to detect and record changes. A programmer may also initiate a scan.

The features of the thesaurus tool described in this section are based on cross-reference information also found in other programming environment tools like source code analysers and data dictionary tools [Bourne 1979, IBM 1980, DEC 1989, SoftwareAG 1990] except that the thesaurus tool spans *all* the languages used to build the *whole* persistent application system, its user interfaces and its databases. The next section describes how the thesaurus tool focuses on the issue of change management.

2.3 Change Management

In order to study the nature of changes to the schema (and other container types) the *Versions_Thesaurus* relation was introduced. *Versions_Thesaurus* is like the *Thesaurus* relation but for two added fields that specify whether a name has been added or deleted and the date of the incident (Figure 2.4). A change to the name of for example a relation has been registered as one deletion and one addition. It is generally impossible for a tool to distinguish between a rename and a deletion followed by an addition without any user provided information. If the structure of the relation changes as well (fields added, deleted or changed), it is also a semantic problem to decide whether the same relation has been modified or a new one has been created. So, a rename of a field or relation is registered as one deletion and one addition, whereas a change to the type of a field is not captured in the thesaurus at present.

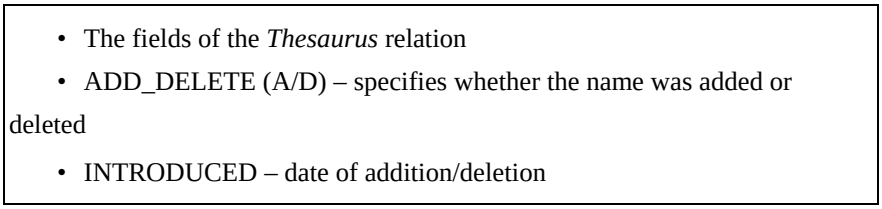

Figure 2.4: The Versions_Thesaurus *relation*

In order to find the effects of schema changes, the *Query_Dictionary* relation was introduced which describes *direct* correspondences between fields of the relations and datums used in the Display Language or Hippo programs (Figure 2.5). This information can generally not be inferred from the *Thesaurus* relation.

- RELATION_NAME
- FIELD_NAME
- QDFUNCTION_NAME – a name of a query or update function
- DATUM_NAME

Figure 2.5: The Query_Dictionary *relation*

The thesaurus interface has one window displaying information from the *Thesaurus* relation and another window displaying information from the *Query_Dictionary* relation. The interface also provides three "Change to X" buttons which execute queries for finding the name occurrences possibly affected by changes to a relation, field, query or update function. For example, if the query dictionary table of the interface contains some entries (a result of another query), a user can select (say) an occurrence of a field name and then press the "Change to Field" button. Figure 2.6 shows an example where the field BED_NO of the BED relation has been selected.[4] In the query dictionary window, all entries having the actual field name are displayed. The thesaurus window displays all occurrences[5] of all datums corresponding to this field and all queries and update functions containing occurrences of the field. There is also a similar "Change to Relation" button. Though not a schema change, the "Change to QDfunction" button performs a query which finds all scripts and programs using a selected query or update function and all relations and fields referred to within this query or update function.

[4] Figure 2.6 is only a sketch of the actual screen showing the functionality. The real system is implemented using colour-graphics on high resolution workstations (the screen dumps are unreadable).

[5] In this paper *occurrence* denotes an occurrence of an identifier – a *name* of a datum, field, etc., not its definition or value.

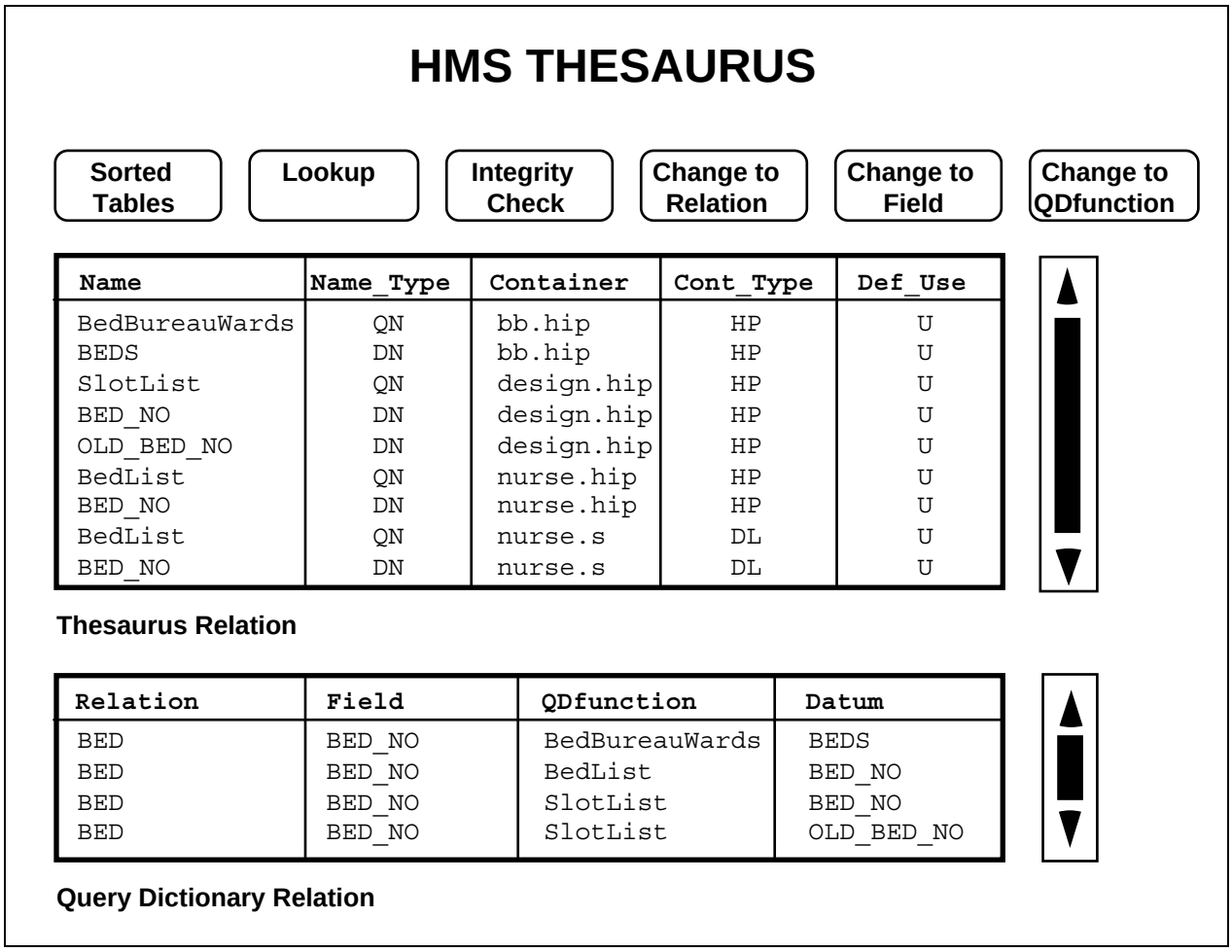

Figure 2.6: The Thesaurus interface

The thesaurus tool indicates where changes *might* have to be done – it does not actually perform any changes or conversions itself. The specification and construction of such tools are an issue for further research.

3 Results of Measuring the HMS System

This section presents measurements of the changes to the HMS schema and measurements of the consequences of such changes. The period for the study started in June 1990 and continued until December 1991. Initially, the HMS system was analysed every fortnight, but due to repetitive changes to the development environment and because the author was not present to instantly adapt the tool to these kind of changes, sustaining this frequency proved impossible (see below).

All measurements until November 90 were in the development period. Field trials began in November 90. During the year from November 90 to November 91 the HMS system

development continued with operational use in one hospital beginning in May 91.[6] By December 91 HMS was running in several hospitals. The project team grew from six to thirteen people during the period of investigation.

3.1 Evolution of the HMS Schema

During the period of study, the number of relations increased from 23 to 55 (139% increase) and the number of fields increased from 178 to 666 (274%). However, what is more interesting than this considerable growth in size, is that every relation has been changed. At the beginning of the development almost all changes were additions. After the system provided a prototype and later went into production use, there was not a diminution in the number of changes, but the additions and deletions were more nearly in balance.

	Relations			Fields		
Date	**Added**	**Deleted**	**Current**	**Added**	**Deleted**	**Current**
22/6/90			23			178
6/7/90	6	0	29	103	0	281
20/7/90	13	0	42	78	0	359
3/8/90	1	-1	42	9	-15	353
17/8/90	18	0	60	97	0	450
Oct-90	3	-23	40	52	-126	376
Nov-90	47	-40	47	528	-376	528
Nov-91	40	-28	59	550	-290	788
Dec-91	20	-24	55	229	-351	666
Total	148	-116		1646	-1158	

Table 3.1: Added and deleted relations and fields in the HMS schema

Table 3.1 shows the development for the relations and fields. (A diagrammatic interpretation is given in Figures 3.1 and 3.2, respectively.) The number of deleted relations and fields appears as a negative value, so the *Current* value is the previous *Current* value plus the values of the *Added* and *Deleted* columns. *Added* and *Deleted* include both fields explicitly added to and deleted from a relation *and* fields added and deleted implicitly as a part of an addition or deletion of a relation. Most changes to the fields are such implicit changes. However, there are a substantial number of explicitly added and deleted fields as well. For example, of the 20 relations found in both the November 90 and November 91 schemata, only 4 have unchanged structure (the fields remained

[6] The operational system concerned the management of in-patient information. Many of the changes were the result of improvements to this system, changed requirements by government (the minimum data set) and the development of an out-patients system due for delivery in April 1992.

the same). During the period of examination, a total of 148 relations and 1646 fields have been added, whereas respectively 116 and 1158 have been deleted. That is, there have been 28% (relations) and 42% (fields) more additions than deletions.

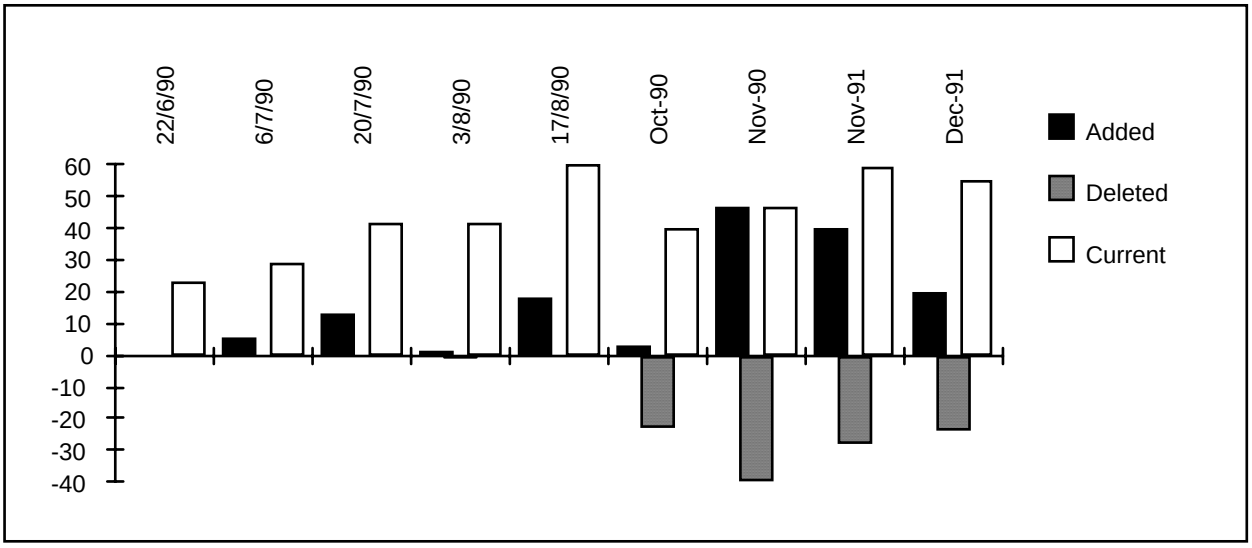

Figure 3.1: Change history of the relations

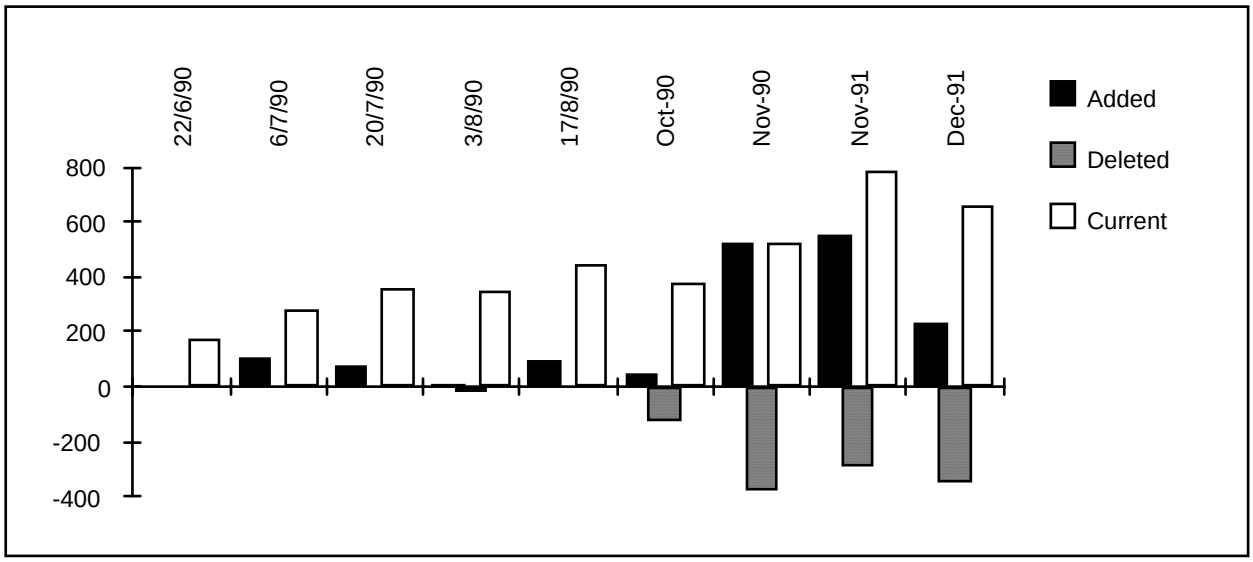

Figure 3.2: Change history of the fields

As mentioned, rename of a field or relation and changes to the type[7] of a field are not captured by the automatic measurements. However, a visual check on the November 91 and December 91 schemata found that there was only one rename of a relation where the relation's structure was unchanged, that 3 relations were vertically factored and that in one case 2 relations were joined together. The rest were “pure” additions and deletions. Regarding the fields, there were 18 renamings, 4 changes of unique/non-nulls, 23 changes of length and 4 changes of representation (3 from character to integer and one vice versa), i.e. 31 changes of field type. Respectively 31 and 48 fields were explicitly added and deleted.

In a large-scale project, with many people involved, there will always be different interests and different opinions on how to solve the problems. Changes of the specification, context and customer generate drastic changes to the project. This was for example the case in the HMS project when the November 90 version replaced the October 90 version.

3.2 Consequences of the Schema Evolution

The previous section gives an impression of how significantly the HMS schema changed during the period of investigation. In order to provide a consistent application system, such schema changes have to be propagated to the application code. This necessary change propagation will be discussed in terms of the extent to which programs must be changed (edited) for each kind of schema change. The modification of the Nov-91 schema into the Dec-91 schema will be used as an example when describing the impact on the application code. A presentation of the use of the relations and fields in the Nov-91 version of the HMS system should help understand the example.

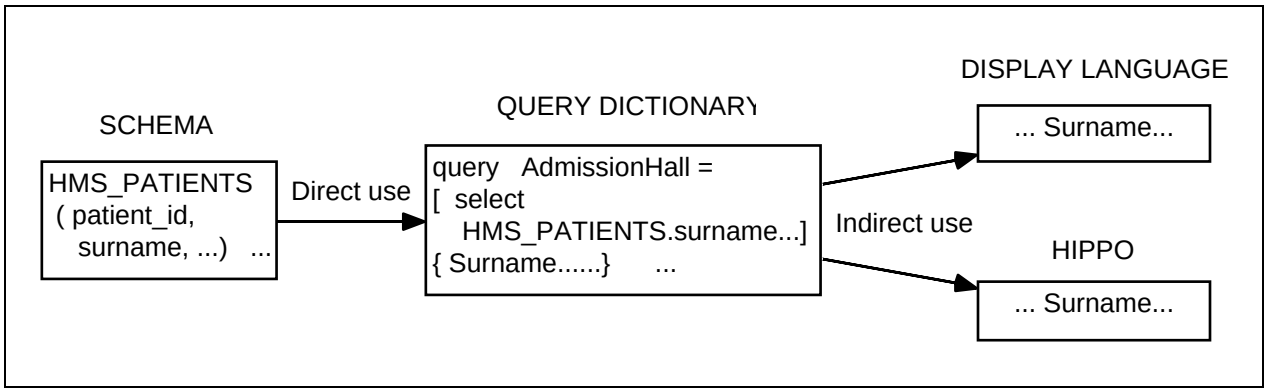

Figure 3.3: Direct and indirect use of relations and fields

Screens, actions, functions, queries, update functions, etc. are all dependent on the schema. The references to relations and fields in the screens and actions are all indirect via the query

[7] A very general interpretation of the type concept is here used which includes the field properties unique, non-nulls, length and representation (*integer, char, date,* etc.).

dictionary. The query dictionary was introduced to absorb change. Its analogy is a traditional view mechanism, but the query dictionary is more general supporting update and allowing interfacing to different DBMSs. Schema changes have direct consequences only for the query dictionary, but in general it is necessary to propagate these changes to the Display Language and Hippo code. For example, if the relation *HMS_PATIENTS* gets a new attribute, *place_of_birth*, the actual values must be entered via a screen (Display Language code). Furthermore, at least one application program should utilise this new information. Figure 3.3 illustrates the direct and indirect use of relations and fields. In the example, the query *AdmissionHall* uses the field *HMS_PATIENTS.surname* whose value is assigned the datum *Surname* which, in turn, is used in Display Language and Hippo code.

Measurement	**Number**	**Min**	**Max**	**Mean**	**Std**	**Sum**
Relations	59	0	101	16.9	27.1	997
Fields	788	0	167	6.6	14.2	5181
Fields grouped by Relation	59	0	795	87.8	178.3	5181

Table 3.2: Direct use of relations and fields in the query dictionary

Table 3.2 describes the direct use of relations and fields in the query dictionary. The first measurement, “Relations”, shows that among the 59 relations there is at least one which is never used (*Min*) and at least one other used 101 times (*Max*). The average is 16.9 (*Mean*), and the total number of times a relation name appears in the query dictionary is 997 (*Sum*). The standard deviation (*Std*) is high because most of the use is represented by only a few relations.

Both the “Fields” and “Fields grouped by Relation” measurements describe use of the fields. The extra information obtained by introducing “Fields grouped by Relation” is that the field statistics are related to the associated relation. For example, the maximum value 795 in row of “Fields grouped by Relation” indicates that there is at least one of the 59 relations which has in total 795 occurrences of its fields. An analysis of the raw data reveals that the fields of 3 relations constitute 45% of the use which implies a high standard deviation. The maximum number of occurrences for a field is 167, whereas the average is 6.6. The total number of field occurrences in the query dictionary is 5181.

Measurement	**Number**	**Min**	**Max**	**Mean**	**Std**	**Sum**
Fields	788	0	193	5.0	14.0	3946
Fields grouped by Relation	59	0	661	66.9	152.5	3946

Table 3.3: Indirect use of fields in Display Language and Hippo code

Table 3.3 shows the indirect use of fields in the Display Language and Hippo code. These measures have been obtained by:

i) finding all correspondences between fields and datums in the queries and update functions,

ii) finding all the queries/update functions[8] and datums used in the Display Language or Hippo code, and by

iii) joining the results of i) and ii) by query/update function and datum.

The 788 fields are on average used indirectly 5.0 times, whereas the measure for fields grouped by relation is 66.9 times. The use of a field and all fields of a relation ranges from 0 to 193 and 0 to 661 occurrences, respectively.

As an illustration of consequences of schema changes, the effect of the modification of the Nov-91 schema into the Dec-91 schema is now described. Figure 3.4 shows that more than one third (36%) of all name occurrences had to be deleted. There were only a few renamings (less than 1%). The consequences of adding relations and fields are difficult to measure, but the minimum number of necessary additions can be estimated to about 10% of the number of existing name occurrences (see discussion below).

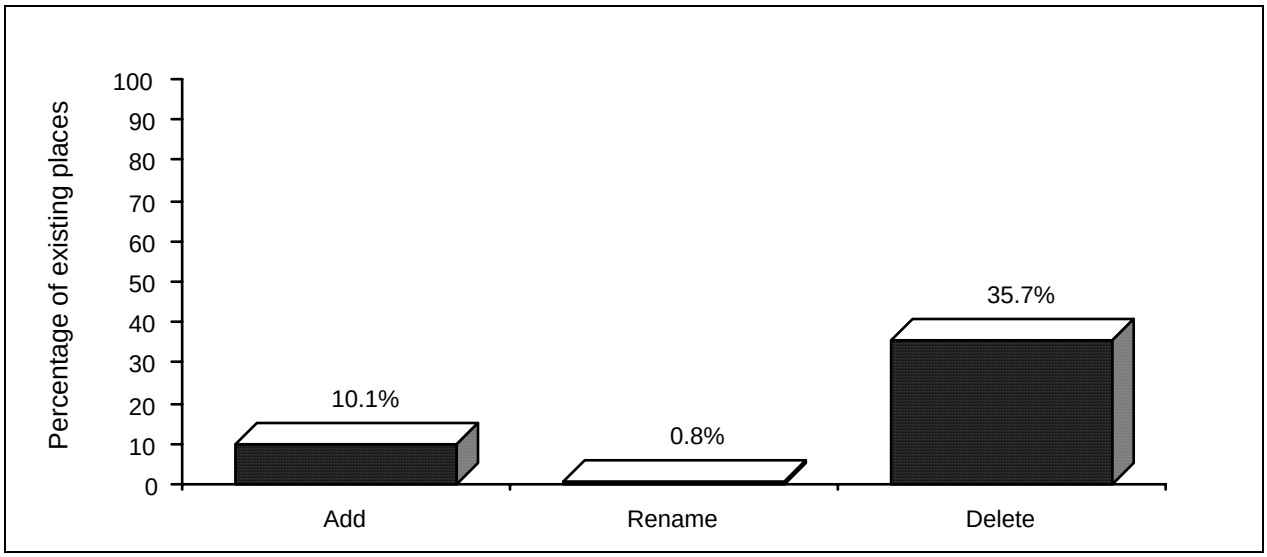

Figure 3.4: Consequences of the December 1991 HMS schema modification

A more detailed description of the consequences is given in Table 3.4 which contains one row for each kind of schema modification. (The number in brackets is the number of occurrences of the named change.) The change consequences are measured in terms of how many *places* which need to be edited for the changed relations and fields. A place is a position in a query or update

[8] A transaction call is here regarded as a call to all its containing update functions.

function where a relation, field or datum name occurs or where a datum name in a Display Language or Hippo program occurs.[9] Duplicates have been removed. That is, the measurements record only one occurrence of a relation, field or datum name in each container. (In the actual code there is about twice as many occurrences.) In Table 3.4 *Query Dictionary* means queries or update functions and *DL or H* means Display Language or Hippo programs.

	Query Dictionary			DL or H	
Operation (occurrences)	**Relations**	**Fields**	**Datums**	**Datums**	**Total**
Add relation (19)	38	360	360	360	1118
Add field (31)		62	62	62	186
Rename relation (1)	8				8
Rename field (18)		128			128
Delete relation (23)	268	1555	628	1370	3821
Delete field (48)		351	151	156	658
Total (140)	314	2467	1201	1948	5930

Table 3.4: Consequences of the December 1991 HMS schema modification

For each added field at least one screen (Display Language code) should collect the new data, and an update function should insert it into the database. Moreover, at least one Display Language or Hippo program should eventually use the new data which also implies a new or modified query. To collect and use the fields of an added relation, the argument above implies that the relation name must be included in an update function and query as well. So, the names of the 19 added relations in the Dec-91 schema[10] must be inserted into the query dictionary at least 38 times. These relations have 180 fields implying that minimum 360 places for the fields and 360 places for the corresponding datums must be edited in the query dictionary and at least the same number of datum names in the Display Language or Hippo code. It is generally impossible for a tool to detect places affected by additions. Human intervention is required.

The renaming of the single relation and the 18 fields cause at least 8 and 128 places to require editing. There is not necessarily any effect on the Display Language or Hippo code because the name change may be absorbed in the query dictionary. However, if the intention is that new field names should be propagated to the corresponding datums, then 97 datums in the query dictionary and their 112 uses in the Display Language and Hippo code would also have to be edited (not shown in Table 3.4).

[9] A place could be localised by for example a (line number, word number) pair.
[10] Table 3.1 shows 20 added relations (not 19) because the single renamed relation is registered as one deletion and one addition by the thesaurus tool.

An examination of Table 3.2 reveals that removing a relation will on average affect 87.8 field occurrences in the query dictionary. In the best case, no field occurrences will need to be edited, but 795 in the worst case. The average number of field occurrences of the 23 actually deleted relations is 67.6, indicating that these relations are used less than average. The consequences of the deletions, however, are still significant. The deleted relations cause 268 removals of the relation names and 1555 removals of the name of their fields. These field names correspond to 628 datums[11], which have 1370 occurrences in the Display Language or Hippo code. In summary, the number of places affected by the deletion of the 23 relations is 3821.

In addition to the changes described above, some new update functions and queries will generally be needed which may be referenced in the Display Language or Hippo code. However, the query dictionary may absorb such changes because the same update functions and queries can operate on new relations and fields with only internal changes. That is, their references in Display Language and Hippo code may be unchanged. So, introducing a query dictionary is one means of alleviating the consequences of schema changes.

4 Problems of Measuring Schema Evolution

The thesaurus tool was installed to measure the changes to the HMS schema and its consequences over the 18 month period from June 1990 to December 1991. However, in addition to the changes to the HMS schema and application programs, the system structure and development environments also changed significantly (mainly to cope with the growth of the system). The thesaurus tool itself had to be changed correspondingly. The kinds of change were:

- Completely new structure and names of directories and change to file name conventions.
- Changes to the support software (operating system, DBMS, version control systems, etc.).
- Changes to the application programming languages, like modified syntax and extended run-time library (the query dictionary language, Display Language and Hippo language were all changed during the period of investigation).

Keeping the continuity of the observations may prove difficult due to such changes. (As mentioned, this was the reason for the different time intervals shown in Table 3.1.) Major changes to the languages used may complicate comparisons between versions of the application system though such changes may be unusual in a typical programming environment. However, anybody attempting to carry out similar experiments or build equivalent tools would certainly need to cope with changes in data structures and new versions of support software. In the HMS system the program directories were reorganised without notifying the thesaurus tool. This excluded several programs from the analysis for a short period of time. Another failure was that the program for unloading the thesaurus data from the database was not recompiled when a new version of Oracle was introduced. The result was that no data was unloaded. The tool then

[11] Not all fields in a query or update function correspond to a datum. There are 0.62 datums per field on average.

assumed (wrongly) that the thesaurus relations were empty, and the subsequent test for change detection was invalidated. Therefore, thesaurus tools need to be subject to the same change control mechanisms as the rest of the system under study.

Completely automated collection of change data seems impossible. Therefore, in order to collect reliable measurements of a real-world system the application development people on the site must have the time and interest in co-operating in the experiment. A problem is to convince them that the data collection is worth the investment. This problem may not be so great if the change measurement and management tools were closely integrated with the programming environment.

5 Conclusions and Further Work

Managing the consequences of changes to application systems is a dominant activity in the software industry. In order to provide measurements of the kind and scale of these changes, a relational database application, a health management system, was studied in depth during an 18 month period. The study reveals that schema changes are significant both in the development period *and* after the system has become operational. The main results were:

- Number of relations: 139% increase.
- Number of fields: 274% increase.
- Every relation was changed.
- 35% more additions than deletions.

The consequences of the schema changes on the application programs have also been measured. The results confirm that *change management tools* are needed – at least in the context of advanced and experimental application development such as that measured here.

The measurements were obtained by the thesaurus tool which analyses the database schema and application programs and extracts information about programmer-introduced names denoting relations, fields, screens, actions, queries, update functions, etc. Changes to the set of occurrences of these names are also recorded. In particular, the thesaurus tool provides information about how many screens, actions, queries, etc. may be affected by a possible schema change. In such a way it can be used for estimating the consequences of possible schema changes. Some of the statistics presented and the thesaurus raw data reveal possibilities concerning optimisation strategies.

Most of the recent research on schema evolution has focused on object-oriented databases. Ideas for managing the impact of schema changes on the schema itself (class hierarchy) and on extensional data (objects) have been implemented. Managing the consequences on application programs (methods) proves to be a more complex issue. The results reported in this paper were based on the use of a relational DBMS and confirm that change to database schemata is an important issue independent of the data model of the actual application. A long term goal is

therefore to identify properties related to change consequences that are independent of data model and application.

We all know that there is a significant number of changes going on, but we should now start quantifying them. The schema modification measurements described in this paper are a step in this direction. Although we believe that others must have studied similar systems, we have been unable to find reports on corresponding measurements. The extent and sort of change may differ from our study. In general, change statistics from other projects should be collected, enabling systems in various application domains to be compared in a bigger study.

The causes of change may also vary from system to system.[12] These causes, however, are another research issue and are regarded as irrelevant in our context. The key point is that our measurements of a real, industrial system confirm that designers of tools for the management of large, long-lived systems involving databases must address the problem of changes to schemata. The traditional view of first defining a (fixed) schema and thereafter developing the dependent application programs has proved inappropriate.

The thesaurus tool directs the programmer to places in programs which may need additional changes. At present, the changes are performed by hand. A research issue is how to provide a general change management model in which (some of) those changes can be automated. The measurements show that addition is the most frequent kind of change, followed by deletion. Renaming does not occur so frequently and may be absorbed by organising the software appropriately (though a model for automatic renaming should be relatively simple). It is generally impossible to automate additions – human intervention is required, but a tool may suggest alternative places. In contrast, a model for automatic deletion is conceivable.

The development of a thesaurus-based software information tool (TSIT) [Sjøberg 1992] establishes a platform for research on change management in the context of Napier88 – a strongly typed, persistent programming language. In TSIT the ideas and principles behind the HMS thesaurus tool have been further developed, and Napier88 provides an appropriate platform since application programs can be stored as values within the database and as such are susceptible to manipulation by change management software. In addition to change advisors indicating necessary propagation of type changes, we have also started developing other tools based on TSIT. The TSIT information can be utilised in a consistency check tool that checks if all declared types or values are ever used within an application, whether a persistent value declared in a program actually exists in persistent store (avoiding run-time errors), etc. Another TSIT dependent tool, called *EnvMake*, is meant to replace the use of Make [Feldman 1979] and script files and is tailored to the construction and maintenance of persistent systems. Semantics of programs and persistent store extracted from TSIT enables EnvMake, among other things, to help organise the interaction between programs and environments in persistent store.

[12] In the HMS case considerable investment (much in excess of coding costs) went into design and planning. Changes were still encountered due to changing organisational needs, changing regulations and the addition of major new subsystems.

Acknowledgements

The author would like to thank Malcolm Atkinson for suggesting the thesaurus and for advice as the work proceeded. Thanks also to Ray Welland for his support of this work and to Paul Philbrow, Richard Cooper, Phil Trinder, Ivan Tabkha and Niall Jackson for useful comments on earlier versions of this paper. Andy England, Don More, Duncan Batey, Brenda Selwyn and other staff at Perihelion Software Ltd. (PSL) have been co-operative and helped carrying out the work. The generosity of Dr. Tim King of PSL, in permitting access to project and providing support for the extended visits to PSL, is very much appreciated.

The author holds a fellowship from the Norwegian Research Council for Science and the Humanities (NAVF). Some of the work has been associated with the FIDE project (ESPRIT Basic Research Action 3070).

References

[Atkinson 1991] Atkinson, M.P., FIDE Course, Section 2.5, version 1.2, FIDE – ESPRIT Basic Research Action Project 3070, May 1991.

[Banerjee *et al.* 1987] Banerjee, J., Kim, W., Kim, H.-J. and Korth, H.F., "Semantics and Implementation of Schema Evolution in Object-Oriented Databases", Proceedings of the ACM SIGMOD 1987 Conference on the Management of Data, San Francisco, CA, 27th-29th May, pp. 311-322, 1987.

[Bourne 1979] Bourne, T.J., "The Data Dictionary System in Analysis and Design", ICL Technical Journal 1(3), pp. 292-298, Nov. 1979.

[Chen 1976] Chen, P.P., "The Entity-Relationship Model – Toward a Unified View of Data", *ACM TODS*, Vol. 1, No. 1, pp. 9-36, 1976.

[Chikofsky and Cross 1990] Chikofsky and Cross, "Reverse Engineering and Design Recovery: A taxonomy", *IEEE Software*, January 1990.

[Clifton 1990] Clifton, N., *Display Language Documentation*, October 1990.

[Codd 1970] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM, 6, 13, 377-387*, June 1970.

[Corbi 1989] Corbi, "Program Understanding: Challenge for the 1990s", *IBM Systems Journal Vol. 28 No 2*, 1989.

[DEC 1989] *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual*, Digital Equipment Corporation, 1989.

[England and Selwyn 1990] England, A. and Selwyn, B., *Hippo Language Guide*, November 1990.

[Feldman 1979] Feldman, S.I., "Make – A Program for Maintaining Computer Programs", Software – Practice and Experience, 9, 255-265, 1979.

[IBM 1980] *DB/DC Data Dictionary General Information Manual*, IBM, 1980.

[Joseph *et al.* 1989] Joseph, J., Thatte, S., Thompson, C. and Wells, D., "Object-Oriented Database Workshop in OOPSLA'88", *SIGMOD Record, Vol. 18, No. 3*, September 1989.

[Kim and Chou 1988] Kim, W. and Chou, H.T., "Versions of Schema for Object-Oriented Databases", *Proc of 14th VLDB Conference*, Los Angeles, 1988.

[Lerner and Habermann 1990] Lerner, B.S. and Habermann, A.N., "Beyond Schema Evolution to Database Reorganisation", *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, October 1990.

[Morrison *et al*. 1989] Morrison, R., Brown, F., Connor, R. and Dearle, A., *The Napier88 Reference Manual*, Universities of Glasgow and St. Andrews, PPRR-77-89, 1989.

[Parikh and Zvegintsov 1983] Parikh and Zvegintsov, "The World of Software Maintenance", *Tutorial on Software Maintenance*, Parikh and Zvegintsov, eds., CS Press, Los Alamitos, CA, 1983.

[Penney and Stein 1987] Penney, D.J. and J. Stein, "Class Modification in the GemStone Object-Oriented DBMS", *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, October 1987.

[Putnam 1982] Putnam, L.H., "Software Cost Estimating and LifeCycle Control", *IEEE Catalog*, 1982.

[Sjøberg 1991] Sjøberg, D., *The Thesaurus – A Tool for Meta Data Management*, Technical Report FIDE/91/6, ESPRIT Basic Research Action, Project Number 3070---FIDE, February 1991.

[Sjøberg 1992] Sjøberg, D., *Measuring Name and Identifier Usage in Napier88 Applications*, FIDE Technical Report FIDE/92/37, ESPRIT Basic Research Action, Project Number 3070---FIDE, 1992.

[Skarra and Zdonik 1987] Skarra, A.H. and Zdonik, S.B., "Type Evolution in an Object-Oriented Database", Shriver, B.S. and Wegner. P. (Eds)., *Research Directions in Object-Oriented Programming*, MITP, Cambridge, MA, Computer Systems, pp. 393-415, 1987.

[SoftwareAG 1990] *The Predict Reference Manual Version 3.1*, Software AG, Germany, 1990.

[Zelkowitz 1978] Zelkowitz, M.V., "Perspectives on Software Engineering", *ACM Computing Surveys, Vol. 10, No.2*, June 1978.