Elementary Structures in Entity-relationship Diagrams as a Diagnostic tool in Data Modelling and a Basis for Effort Estimation

Geoffrey J. Kennedy

Department of Information Science
University of Otago, Dunedin, New Zealand
Email: gkennedy@infoscience.otago.ac.nz

Abstract

Elsewhere Kennedy describes three elementary structures to be found in entity-relationship diagrams. Here, each of these structures is considered in the context of a transaction processing system and a specific set of components that can be associated with the structure is described. Next, an example is given illustrating the use of elementary structures as an analytical tool for data modelling and a diagnostic tool for the identification of errors in the resulting data model. It is conjectured that the amount of effort associated with each structure can be measured. A new approach for the estimation of the total effort required to develop a system, based on a count of the elementary structures present in the entity-relationship diagram, is then proposed. The approach is appealing because it can be automated and because it can be applied earlier in the development cycle than other estimation methods currently in use. The question of a suitable counting strategy remains open.

Keywords

Data modelling, Design Tools and Techniques, Entity-relationship model, Software Metrics

ELEMENTARY DATA STRUCTURES IN ENTITY-RELATIONSHIP DIAGRAMS

Figure 1 shows the three elementary structures to be found in entity-relationship diagrams as identified by Kennedy (1992). **A**, **B** and **C** represent three entities[1] each containing two or more attributes. The attribute **a** is the key of entity **A** (indicated by underline), **b** is the key of **B** and **c** of **C**. It is assumed that the model will be implemented by means of a relational database whereby relationships between entities will be represented by means of appropriate foreign keys (indicated by the use of parentheses). The three diagrams given exhaust the distinct possibilities involving non-cyclic arrangements of three such entities. Kennedy labels the three possibilities as *Type I*, *Type II* and *Type III* structures respectively.

In this paper it will be demonstrated that when considered in the context of a transaction processing system each structure has associated with it certain system components that will be required for any such information system. It will be further demonstrated that the recognition of these structures proves useful as an aid in data analysis and in the identification of errors in the resulting data model. Finally it will be argued that since each of these components absorbs a measurable amount of effort, the possibility presents itself of using counts of elementary structures as the basis for the estimation of effort required to develop the information system.

IMPLEMENTATION CHARACTERISTICS OF ELEMENTARY STRUCTURES

Each of the elementary structures shown in Figure 1 can be identified with familiar real world situations. By considering each structure as part of a typical transaction processing system it can be seen that they will, in general, have associated with them a well-defined set of system

1. Strictly these should be referred to as ‘entity classes’ but the term ‘entity’ will be used interchangeably here.

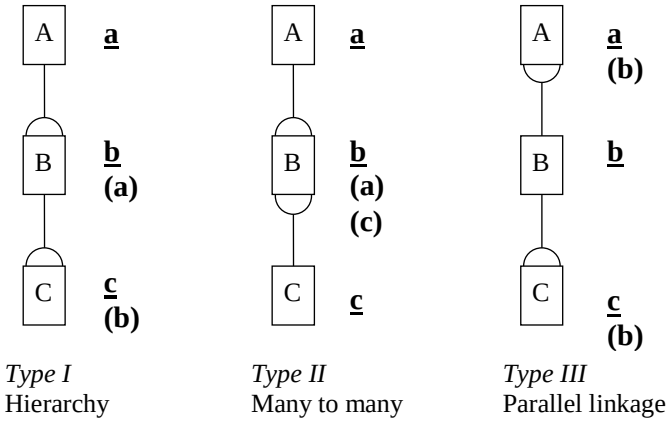

Figure 1: Elementary structures in entity-relationship diagrams

components necessary for the capture and retrieval of data contained in the system database. In some cases there will also be some specific data processing requirements. The implementation characteristics of each of the structures in turn are described in the following examples.

Type I Structure

The Type I structure can be recognised as a typical *hierarchy*, in which each 'child' occurrence is associated with one and only one 'parent' occurrence. One example, shown in Figure 2, is the familiar triple CUSTOMER - ORDER - ORDER_LINE data model. Other easily recognisable examples are:

CITY - SUBURB - STREET
DIVISION - DEPARTMENT - EMPLOYEE

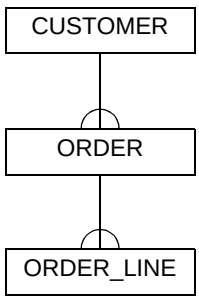

Figure 2: Type I structure - simple hierarchy

When this structure is considered in the context of a typical transaction processing system it can be seen that in most circumstances the following screens (or forms) will be required.

(i) A file maintenance screen for the parent entity, the CUSTOMER entity in the example given. This screen will be used to create records for new customers and to modify existing occurrences. The screen will require some mechanism for selecting the required parent record, such as a drop-down list box. The maintenance screen may display either a single occurrence, or allow several records to be viewed at a time.

(ii) A screen to list all dependent records of a selected parent. In the example given it will display all ORDER records belonging to a selected customer (see Figure 3). It will usually also need to access the ORDER_LINE table in order to display totals[1] for each order. Implementation of the parent-child screen illustrated, or 'one-to-many'

as it is sometimes called, is specifically supported in many system development packages. Alternatively, the same functionality may be implemented as two related screens, one for ORDERs for example, and the other for associated ORDER_LINEs, sometimes referred to as a subscreen. In any case the subscreen will receive values of the parent record passed to it from its master screen. It will also need to be able to scroll or in some way display multiple dependent records page by page.

(iii) A screen permitting each child of the parent record to be 'exploded' to display its dependent records. This can be accomplished, for example, by double clicking on the record selected. In the example, details of Order No. 46 may be viewed by double clicking on the required record in the parent screen (Figure 3) resulting in the screen display given in Figure 4. Once again, parent record values will be passed from the master screen to the subscreen.

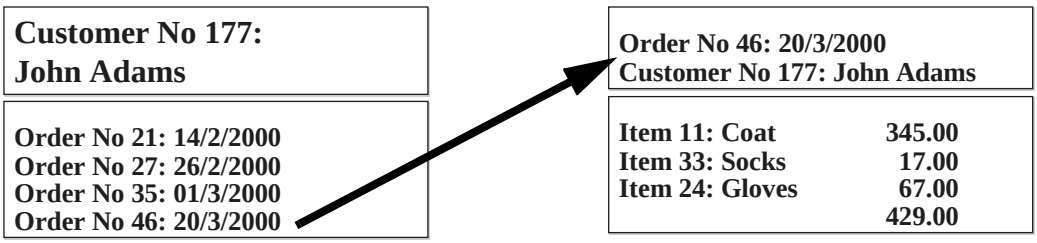

Figure 3: CUSTOMER - ORDER screen

Figure 4: ORDER - ORDER_LINE screen

In addition to the demands of screen design most information systems also have user requirements for reports. These can be either screen based or paper based. Some users demand even simple listings of their data, for example an alphabetical list of customers, but effective reports usually involve some sort of summarising, aggregation or exception reporting. For the orders example we might expect an exception report for customers with unfulfilled orders. In general, the information contained in a report differs from that possible in a screen (or form) because advantage can be taken of a greater variety of possible data processing options.

For the Type I structure report generation is quite straight forward in terms of data retrieval. In most cases data is retrieved starting from a particular parent record, working down the hierarchy. Joins over each foreign key in turn result in the formation of a data complex containing all the related data necessary for reporting. The amount of effort required will depend on the report generation capabilities of the development tool and/or database management system in use, but in principle the task is straight forward. It is also possible to begin retrieval with a specific occurrence of the 'middle' child entity, here for example a specific order, from which the single parent and multiple dependent records can be found. Again, this is straight forward.

Most of the effort involved with reports is in the layout design. Whilst crucial from the point of view of the user, it is not affected by the underlying data structure and will be the same no matter what elementary structure is involved.

Type II Structure

This structure reflects a *many to many* relationship between entities **A** and **C** in Figure 1. The entity **B** is sometimes referred to as an *associative entity.* In most cases the entity **B** can be associated with some real world object or event, but it may be an artificial entity, required by the relational model simply to implement the many to many association. So far as implementation is concerned, these two situations are indistinguishable, and so do not require distinct

1. The total for each order is not stored, since it can be derived by summing the detail lines.

treatment. The familiar ORDER - ORDER_LINE - PRODUCT data model, shown in Figure 5, is one example. An important feature of this structure is its symmetry. Unlike the Type I structure described above, access to records from either direction is usually required. Other obvious examples are:

STUDENT - ENROLMENT - COURSE
EMPLOYEE - ASSIGNMENT - PROJECT

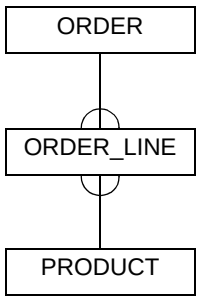

Figure 5: Type II structure - many to many association

In a typical transaction processing system containing a Type II structure it is probable that the following screens will be required.

(i) A file maintenance screen for *one* or *both* of the parent entities. In the orders example, a file maintenance screen will be required for the PRODUCT entity only, since ORDER and ORDER_LINE records will normally be created at the same time as transactions. However, in the other two examples given, maintenance screens will be required for both parents, that is for STUDENTs and COURSEs and for EMPLOYEEs and PROJECTs respectively. As in the case of the Type I structure, file maintenance screens will be used to create new records or to modify existing ones, displaying either single occurrences or several records at a time.

(ii) A 'parent-child' screen to append or list dependent records of the first parent. In the orders example it will be identical to the screen in the Type I example for displaying ORDER_LINE records belonging to a selected ORDER (compare Figure 4 and Figure 6). As before, parent data will be passed to the subscreen.

(iii) A second 'parent-child' screen displaying child records of the other parent record. In the orders example this will show all orders containing a selected PRODUCT (see Figure 7). As for all such cases, a mechanism for selecting a particular parent record (product) is required and data is passed to the subscreen.

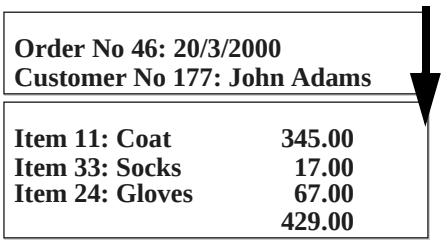

Figure 6: First parent ORDER - ORDER_LINE

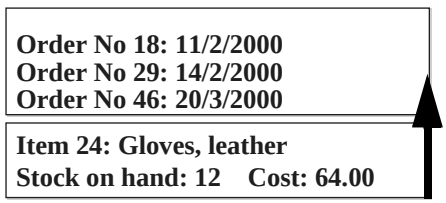

Figure 7: Second parent PRODUCT - ORDER_LINE

Just as for the Type I structure, report generation with Type II structure is straight forward in terms of data retrieval. Once again, joins over the foreign keys result in data complexes con-

taining all the data necessary for reporting. Some report generators simplify matters by permitting joins in either direction, but it is not a crucial factor in the production of reports.

Type III Structure

This third elementary structure (see Figure 1) occurs in situations where a parent entity **B** is associated with more than one dependent entity, each of which can have multiple occurrences. Occurrences of entities **A** and **C**, whilst separately related to **B**, are not in any way related to each other. It is sometimes referred to as a *parallel linkage*. One example, shown in Figure 8, is the case of a CUSTOMER at a bank, for whom we wish to record a history of DEPOSITs and WITHDRAWALs. The point to remember is that there is no relationship implied between a particular deposit record and any withdrawal. It is this feature which distinguishes the Type III structure.

The structure is probably more easily recognised when arranged as shown in Figure 9, showing the two 'child' entities, each dependent on the same 'parent' but independent of each other. Other possible examples are:

TRAINING - EMPLOYEE - LEAVE
RECEIPT - PRODUCT - SALE

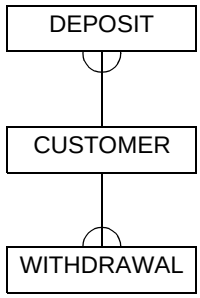

Figure 8: Type III structure - parallel linkage

Screen requirements for a transaction processing system containing a Type III structure are similar to those described already for the Type I (hierarchical) structure.

(i) A file maintenance screen for the parent entity. In the example given this would again be the CUSTOMER table. As in previous cases this form will be used to create records for new customers and to modify existing occurrences.

(ii) A screen for the first of the dependent records, displaying records related to a selected parent. In the bank example illustrated in Figure 9 this screen would display DEPOSIT records belonging to a selected customer. It will be similar in appearance to that shown in Figure 3. As before, data values will be passed from the master screen and the subscreen must be able to display multiple child occurrences page by page.

(iii) A second screen, similar to (ii), will display occurrences of the other dependent entity, in this case WITHDRAWALs.

Besides the screen requirements described above, the Type III structure has associated with it special data processing requirements necessary for the production of reports. A report containing data from both dependent entities associated with occurrences of parent records in a Type III structure is more complicated than for Type I or Type II structures, for instance, in the bank example, a 'statement' listing deposits and withdrawals for each customer in date order (see Figure 10). Not only are the dependent tables (deposits and withdrawals) unrelated, meaning that they cannot be joined directly to produce the required output, but as well they usually have

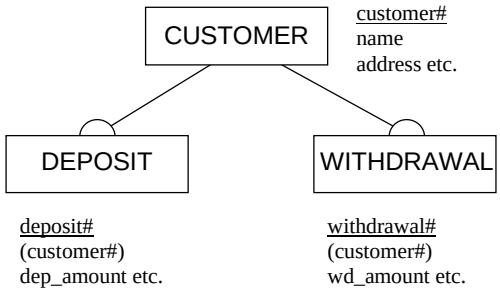

Figure 9: Example of a Type III Structure

Statement

Customer: 983501 Date: 30/10/99
Shane BROWN
12 North East Valley Rd

Opening Balance			4139.00	CR
Date	**Reference**	**Details**	**Amount**	
03/10/98	02322	ATM	140.00	
07/10/98	11994	Cheques	235.00	CR
14/10/98	12020	Salary	3150.00	CR
19/10/98	03102	TT	1600.00	
25/10/98	03152	Autopay Rates	1300.00	
Closing Balance			4484.00	CR

Figure 10: Output from a Type III structure

different structures (i.e. non-homogeneous). Therefore such a report calls for the creation of an intermediate data structure (temporary table) and requires two or more passes through the database to extract the necessary data. The temporary table itself is not necessarily in third normal form. It simply stores the data extracted from each of the tables forming the Type III structure and provides a homogeneous structure suitable for report generation.

The Type III structure can be extended to include any number of ‘child’ entities, each dependent upon the parent, but mutually independent of each other. Each pair of children in turn can be thought of as forming a Type III structure with the parent.

RECOGNITION OF ELEMENTARY STRUCTURES AS A DIAGNOSTIC TOOL

Having characterised the elementary data structures described above, the intention of this paper is now to demonstrate the value of recognising of these structures as an analytical and diagnostic tool in data modelling. In much the same way that an organic chemist can make useful predictions about an unknown compound by recognising the presence of phenol, ketone, aldehyde or other identifiable structures in its molecular structure diagram, consideration of elementary structures in an entity-relationship diagram can assist the system designer in several ways:

(i) it promotes a better understanding of the data model and of the application which it is intended to support,

(ii) it facilitates, even without detailed knowledge of the application, the identification of errors in the data model, such as incorrect, redundant or ambiguous relationships between entities, and

(iii) it draws attention to likely processing complexity in the implementation phase.

At the University of Otago groups of final year students in the information science major tackle information system problems proposed by clients from the local community. Over the past eleven years, working in conjunction with these clients, the author has dealt with over one hundred and fifty transaction processing applications. Consideration of the elementary structures present in entity-relationship diagrams produced by students has been found to be useful, either in confirming that the data model will satisfy the requirements of the application or in detecting anomalies, redundancies or errors. The following example illustrates the approach.

Example: Asset Management Application

A problem proposed by the visual arts department of a local polytechnic concerned the management of bookings and loans of a range of *Assets* belonging to the department, such as 35mm cameras, video cameras, telephoto lenses, tripods, and so on. Assets are classified into various *Categories.* When student *Borrowers* have completed relevant *Units* of study, they are

thereby authorised to borrow items in one or more categories. Staff are also authorised to borrow items. Sometimes equipment is withdrawn from circulation for the purpose of maintenance.

The entity-relationship diagram in Figure 11 shows the initial model proposed by a group of students. The discussion following illustrates how consideration of the elementary structures present in the diagram can help to promote understanding of the application and to identify some design errors in the data model proposed, even without a deep understanding of the application.

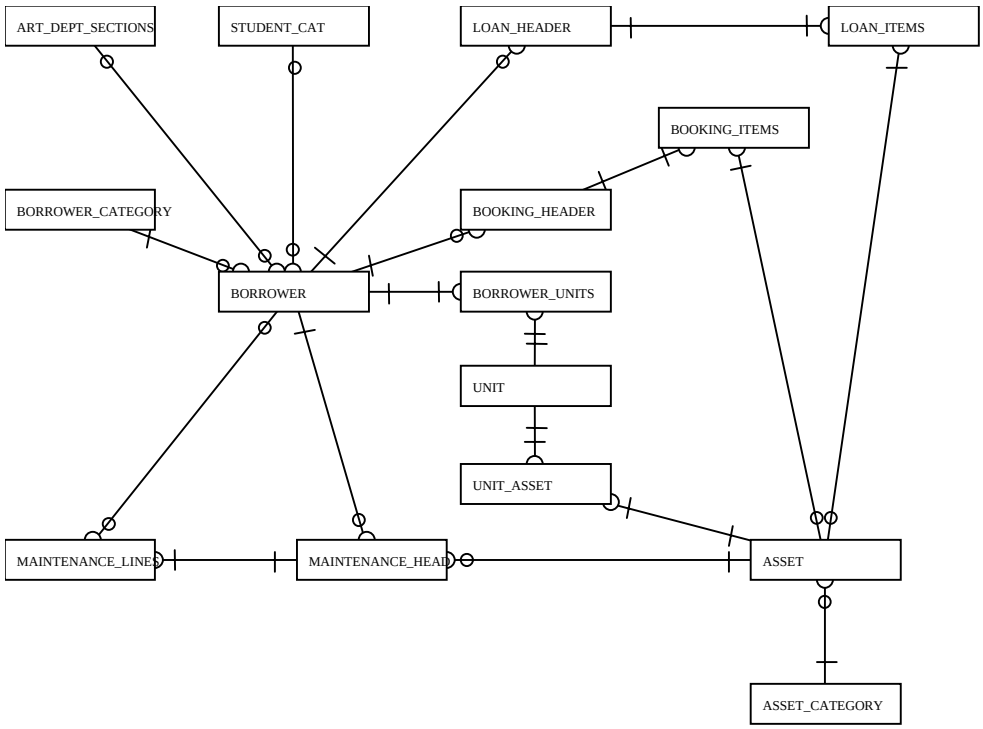

Figure 11: Initial data model proposed for the polytechnic asset management system

(i) The Type I structure (hierarchy) *Borrower - Loan_Header - Loan_Items* clearly duplicates that involving *Borrower - Booking_Header - Booking_Items*. Only one such structure is necessary. The change in status from 'booked' to 'loaned' should be indicated simply by a change in value of a suitable attribute. In fact, further investigation revealed that a 'loan' never involves more than one item, meaning that both structures should be omitted altogether and replaced by the single *Booking_Loan* entity shown in Figure 12.

(ii) The presence of the Type II structure *Units - Unit_Asset - Asset* suggests a many to many relationship between *assets* and *units*. In fact, students become eligible to borrow a whole *category* of assets by virtue of a completed *unit*, so the relationship is between *Unit* and *Asset_Category* as shown in Figure 12. The associative entity *Unit_Asset_Cat* has been introduced to reflect this relationship.

(iii) The cyclic structure involving *Borrower - Maintenance_Head - Maintenance_Line* appears confused and contradictory. Since *Maintenance_Line* records in the hierarchy will always be retrieved via a *Maintenance_Head* occurrence, the link between *Borrower* and *Maintenance_Line* is not needed and should be omitted. The simple Type I structure *Borrower - Maintenance_Head - Maintenance_Line* seen clearly in Figure 12 correctly models the situation. In general, the presence of

cyclic structures can lead to ambiguous data retrieval statements and often can be removed.

(iv) The Type III structure *Borrower_Units - Unit - Unit_Asset* warns of likely complexity in producing reports. A similar situation is also seen in the final model with the Type III structure involving *Borrower_Units - Unit - Unit_Asset_Cat*. This means that it is not possible to list directly all the *Unit_Asset_Categories* associated with particular *Borrower_Units*, as requested by the user. Any output requiring data from both of these tables will require two passes and cannot therefore be included in a screen as a dropdown list, for example.

A corrected model addressing each of the issues raised above is given in Figure 12. The diagram also serves to illustrate how appropriate reorganisation in layout can help to emphasise the structures involved, which in turn can aid understanding of the data model and of the application itself.

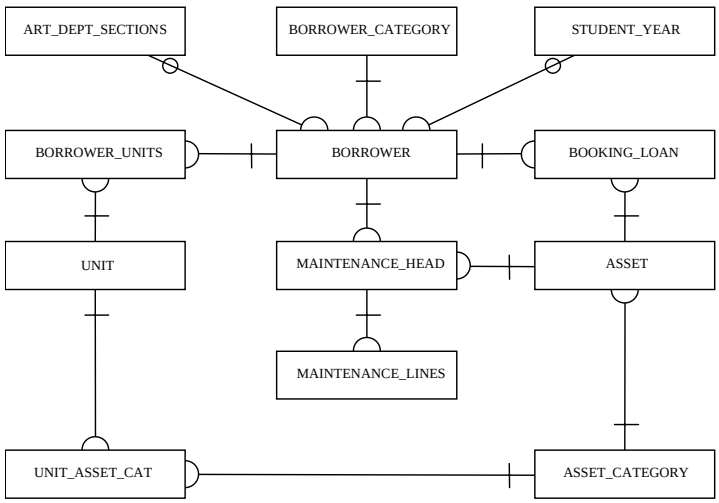

Figure 12: Final data model for the polytechnic asset management system

ELEMENTARY STRUCTURES AS A BASIS FOR EFFORT ESTIMATION

The task of estimating the effort required to develop an information system is an important one but one which always proves difficult. Several approaches have been described in the literature (see for example MacDonell, 1994 or Pengelly, 1995) but when applied the estimates are often poor (Kitchenham, 1992). The approach proposed here draws upon aspects of the 'property-based' approach discussed by Briand *et al.* (1994) and as well aspects of the 'function point' method of Symons (1991). The intention is to base an estimate on factors that can be known as early as possible in the system development cycle. Since the entity-relationship model commonly serves as a starting point for the development of transaction processing systems it provides the earliest possible basis for estimation.

The three elementary structures found in entity relationship diagrams have been identified and characterised. It is obvious that any entity-relationship diagram can be thought of as a network of such elementary structures. When taken in the context of a transaction processing system each of these elementary structures has been associated with a well-defined set of system components, as described in Section 2. It should be possible then to determine experimentally the average amount of effort required to implement each of these basic components, and hence of each of the elementary structures. The actual effort measured will of course depend on the

development tool used and the skill and experience of the developer and so will need to be recalibrated for each new system development situation.

The final task is to count the number and type of elementary structures in the underlying data model and to attribute to each the corresponding amount of effort. The approach does of course make assumptions about the additive nature of effort associated with elementary structures.

One advantage of this proposed method of estimation is that it is more synthetic than the regression approach and therefore relies less heavily on collecting quality effort data. Furthermore, if we assume that the entity-relationship diagram is stored in computer readable form, then it should be possible to automate the method.

The final problem to be addressed is the question of counting the elementary structures. Kennedy has reported some empirical results using what he refers to as an 'exhaustive structure count' which was obtained directly from a computerised entity-relationship diagram (Kennedy, 1996). As an example, structure counts resulting from the application of the exhaustive counting strategy to the entity-relationship diagram in Figure 12 are given in Table 1.

Entities	12
Relationships	12
Type I structures	6
Type II structures	4
Type III structures	3

Table 1: Structure counts for diagram in Figure 12

Applying the exhaustive counting method and using effort data obtained from several groups of students undertaking a range of transaction processing system projects, Kennedy attempted to determine regression coefficients relating the numbers of each type of elementary structure to the total effort expended. The results were disappointing. A major stumbling block was thought to be the poor quality of the effort data. Students were required to keep a 'diary' of time expended on development tasks and, although the response rate was good, inaccuracies in recorded times and obvious inconsistencies between groups tended to render the effort data unreliable.

A second problem identified was with the counting strategy itself. The exhaustive counting method considers each entity in turn, counting every elementary structure in which it participates. Because some entities participate several elementary structures, for example the *Borrower* entity in Figure 12, they are likely to be over represented. The question of a suitable counting strategy remains open. One approach already considered was to make use of so-called 'hot' entities (Kennedy and Hay, 2000). Other suggestions have also been investigated, such as excluding 'lookup' tables from the counting procedure. 'Lookup' tables are those employed simply for data validation. With certain development tools they are very simple to implement and therefore should be treated separately.

CONCLUSION

Kennedy has previously identified three elementary data structures into which any entity-relationship diagram can be decomposed. In this paper it has been demonstrated that in the context of a typical transaction processing system each structure can be associated with particular system components. An example has been given illustrating how the recognition of these elementary structures can be a useful aid to the understanding of a data model and a valuable diagnostic tool for the identification of anomalies or errors in the entity-relationship diagram.

Finally, a novel approach has been proposed for estimation of the effort required for the development of transaction processing systems. The approach assumes that the development effort required for individual system components can be determined experimentally and attributed to corresponding elementary structures in the entity-relationship diagram. The question of an appropriate counting strategy remains open for future research.

REFERENCES

Briand, L., Morasca, S. and Basili, V.R. (1994) Property-based Software Engineering Measurement, *CS-TR-3368, UMIACS-TR-94-119*, University of Maryland, Computer Science Technical Report, November, 28 pages

Kennedy, G.J. (1992) A Systematic Approach to the Specification and Evaluation of an Information Systems Development Methodology, *Ph.D. thesis*, University of Otago, Dunedin, NZ

Kennedy, G.J. (1996) Elementary structures in entity-relationship diagrams: a new metric for effort estimation, *Proceedings*: *Software Engineering: Education & Practice*, Dunedin, January 24-27, IEEE Computer Society, Los Alamitos, 86-92

Kennedy, G.J. and Hay, G.C. (1997) Elementary structures as a basis for effort estimation in information system development projects, *Proceedings*: *Software Engineering: Education & Practice*, Dunedin, January 26-28, IEEE, Los Alamitos, 246-251

Kennedy, G.J. and Hay, G.C. (2000) Identification of Principal Entities in Entity Relationship Diagrams Using Elementary Structure Counts, *Proceedings*: *Eighteenth IASTED International Conference on Applied Informatics*, February 14-17, Innsbruck, Austria

Kitchenham, B.A. (1992) Empirical studies of the assumptions underlying software cost estimation models, *Information and Software Technology*, **34**, April, 211-218

Macdonell, S.G. (1994) Comparative review of functional complexity assessment methods for effort estimation, *Software Engineering J.*, **9**, 3, 107-116

Pengelly, A. *(1995)* Performance of effort estimating techniques in current development environments, *Software Engineering Journal*, September, 162-170

Symons C.R. (1991) *Software Sizing and Estimating Mk II FPA*, John Wiley & Sons, England

COPYRIGHT