Full citation: MacDonell, S.G. (1993) Deriving relevant functional measures for automated development projects, Information and Software Technology 35(9), pp.499-512.
doi: 10.1016/0950-5849(93)90017-W

Deriving relevant functional measures for automated development projects

Stephen G. MacDonell

Computer and Information Science
University of Otago, Dunedin, New Zealand
stevemac@commerce.otago.ac.nz

Abstract

The increasing use of computer aided software engineering (CASE) tools, fourth-generation languages (4GLs) and other similar development automation techniques, has reduced the impact of implementation methods and individual ability on development task difficulty. It has therefore been suggested that measures derived from software specification representations may provide a consistent basis for relatively accurate estimation of subsequent development attributes. To this end, this paper describes the development of a functional complexity analysis scheme that is applicable to system specification products, rather than to the traditional products of the lower-level design and construction phases.

1. INTRODUCTION

Tate and Verner [1] recently suggested that measures extracted from software specifications may be useful in estimating subsequent development attributes, particularly in an automated development environment. They were able to suggest a number of candidate measures, including the number of data model entities and the number of data flow diagram (DFD) processes. Although certainly useful as a first step, these measures were based only on intuitive expectations. This paper represents a progression from the work of Tate and Verner [1] in that a comprehensive set of operational measures covering several specification dimensions and notations is described. Furthermore, selection of the measures was undertaken with the assistance of structured paradigms as a safeguard to ensure that the chosen measures were relevant to the overall goals of a recent empirical study.

The next section of this paper therefore briefly considers the impact that an automated environment has on software assessment, followed by a discussion on the basis of assessment in software specifications. The third section then provides detailed descriptions of the measures chosen for complexity assessment and effort estimation, and of the reasons and assumptions upon which the selection of measures was based. The paper is then concluded with a short summary of the assessment approach and with comments concerning validation.

2. SOFTWARE ASSESSMENT IN AN AUTOMATED ENVIRONMENT

DeMarco [2,3] has suggested that the attributes of an implemented system are directly related to the characteristics of that system's specification. Given the structured and semi-formal nature of several widely used specification models, it has been further suggested that useful and consistent quantitative information relating to system functionality may be derived from these early-phase representations. The utilisation of graphic system models, such as those used in many specification techniques, provides concise yet comprehensive representations of reality; this enables inexpensive analysis of essential system aspects to be performed, without the need to consider excessive low-level details. Graphic functional models can therefore form a useful basis for the extraction of quantitative indications of the scope and complexity of a project. [2,4] With the widespread incorporation of specification modelling techniques within integrated computer aided software engineering (CASE) environments, the automatic extraction and analysis of measures derived from these specification representations is becoming an increasingly likely prospect.

The increasing use of CASE environments and fourth generation languages (4GLs) in the development of commercial software has also helped to make the transformation from requirements representations to constructed systems much less dependent on implementation methods or on the ability of individual developers. [5,6,7] Relative levels of implementation complexity are therefore of less importance when it comes to assessing the ease with which software developed under an automated environment can be constructed and maintained. [8] It is suggested here that specification-based *functional* complexity, that is, the complexity associated only with the required functionality of a system, will instead be the main determinant of these factors.

Commercial software requirements are often specified using entity relationship diagrams (ERDs) and data flow diagrams. [9,10] These representation methods are semi-formal and have been incorporated into a large number of CASE tools. Tate and Verner [1] therefore suggest that they are appropriate representations for early-phase scope analysis. The procedure of detailed requirements analysis, in the commercial domain at least, frequently consists of two areas - data analysis and processing analysis. [11] This approach is illustrated by Benwell et al. [12] in Figure 1, with the implication that both data flow and data structure models be used during the analysis activity. Both perspectives should therefore be considered in an assessment of complexity if this assessment is to be comprehensive.

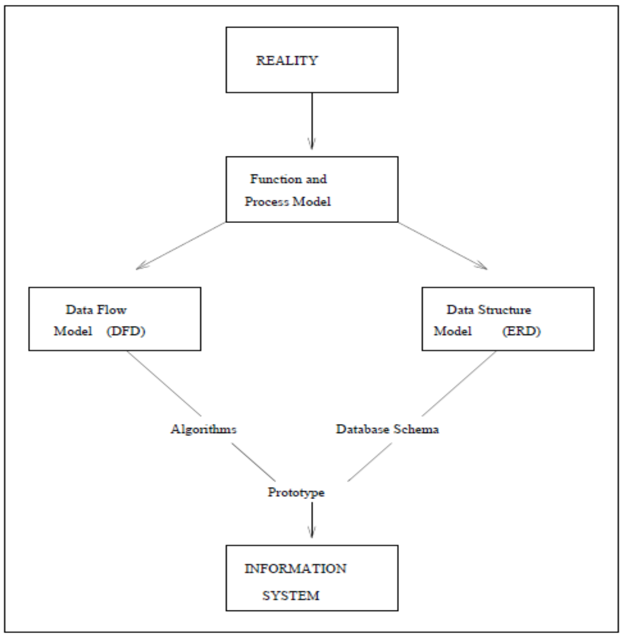


Figure 1. Process and data analysis in software development

Extensive use of the entity relationship (ER) model in determining data requirements is likely to continue in the foreseeable future, especially when it is considered that a large number of automated development environments have adopted the ER modelling convention as the basis for their data repositories. For example, ERMA, Teamwork, IEW/ADW, ProKit Workbench, Data Modeller, Software through Pictures, Excelerator, Blue/60, ER-Designer and IRMA all use the ER model in the derivation of data specifications for software systems. Thus the suggestion that the ER model could be an appropriate early system representation for complexity analysis seems justified.

Given the increasing use of automated assistance in software development, facilities for direct system generation from abstractions such as DFDs should also become more widespread. There are already a large number of commercially available tools that employ data flow modelling techniques (for example, Aut2, Prosa, Teamwork, Excelerator, ProKit Workbench, IEW/ADW). Most include features that assist in the development of robust specifications, for example, the enforcement of consistent element definitions, the detection of duplicate names, process balancing, and co-ordination with the data repository. Processing requirements, as depicted in DFDs, will contribute to the overall functional complexity of a complete specification - any assessment of complexity should therefore consider quantitative aspects of this model.

Three other specification representations are also used extensively in the expression of commercial system requirements. The first is the transaction representation, widely employed in the development of commercial database systems. These systems comprise a number of low-level functions which may be described in terms of the operations that they perform on individual data entities. For example, a process to record a customer payment may read the customer and account entities, then update the account entity. Elementary functions specified using this method can generally be more easily translated into lower-level processing logic than can purely narrative specifications. As this representation combines both data and process requirements it is likely that transaction specifications will provide a sound basis for comprehensive complexity assessment.

Analysing the representation of a system's proposed user interface, the second of the three perspectives considered here, is particularly appropriate for development projects that utilise 4GLs and prototyping methods, as the iterative development of an applicable interface often requires a significant amount of effort in this type of environment. Generally a user interface representation includes examples of the screen and report formats that are required during system operation. As interactive systems, by their very nature, use screen displays, and many transaction processing systems produce reports, a consideration of the complexity of this representation is essential if an overall assessment of complexity is to be obtained.

Another representation perspective commonly employed in commercial system specifications is the functional decomposition hierarchy (FDH). [13] This is a levelled description of the functions that are to be provided by a system, normally incorporating the module calling structure that will subsequently be implemented. Module interaction, at the code level at least, has been the focus of several previous assessment studies (see, for example, Henry and Kafura [14] or Chapin [15]). It is therefore suggested that this representation should also be considered in any comprehensive assessment scheme.

Figure 2 therefore depicts five specification perspectives commonly utilised in commercial systems development. As all five are quantifiable in terms of the contribution that each may make to overall system complexity, each has therefore been considered in the assessment scheme, as described in the next section.

3. ASSESSMENT SCHEME DEVELOPMENT

The Goal/Question/Metric (GQM) paradigm, as developed by Basili and others [16,17] and enhanced by Shepperd, [18] and Bush and Fenton's Classification Scheme [19] were adapted and used to determine the relevant measures for a recent empirical study. [20] The two approaches are similar in that they both employ a decomposition process from overall study goals until elementary data classes or measures essential to achieving the study's objectives have been determined. Figures 3 and 4 depict the use of the two paradigms in the recent study. (More detailed discussion of the use of the GQM and Classification Scheme paradigms in this study can be found in MacDonell. [21])

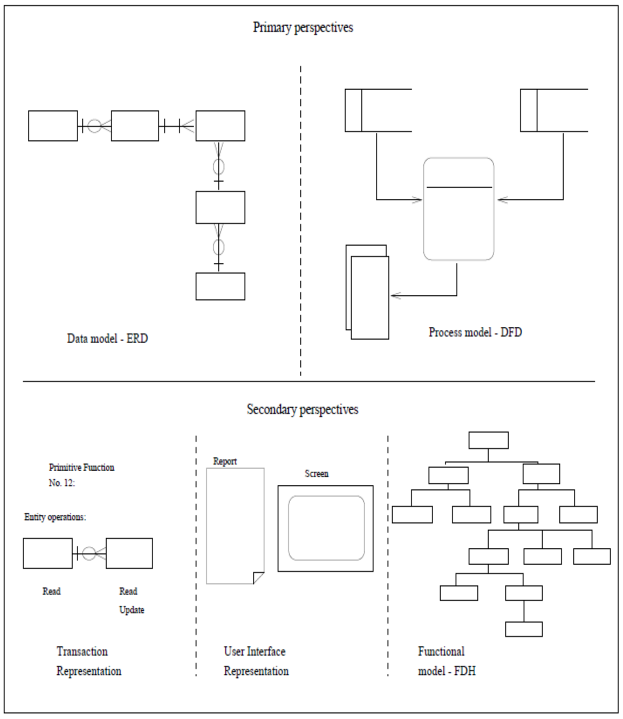


Figure 2. Five perspectives of a system specification

When examined in combination it can be seen that the two figures, 3 and 4, depict all of the data elements needed in the study. The lowest level of Figure 3 shows all of the product measures required to achieve the aims of the project. The necessary project management data elements, however, are not shown on this figure for reasons of clarity. The classification figure, Figure 4, illustrates a similar decomposition of data classes until exhaustive and exclusive project management data elements are specified. Again for reasons of clarity only the characteristic product measures are defined on Figure 4. Thus both figures provide a consistent and comprehensive foundation for the subsequent data collection procedures.

In some cases one of the two process-oriented specification methods considered in the scheme, that is, data flow diagrams and functional decomposition hierarchies, may not be used in a given development project. One of three different data collection strategies would therefore be applicable:

- if both DFDs and FDHs are used then each elementary DFD process should map to a corresponding low-level FDH module - in this case, all five sets of measures should be taken; that is, process model and functional model measures, in association with the transaction, user interface and data model measures;
- if only the DFD is broken down to an elementary level then process model measures only should be recorded, in association with the transaction, user interface and data model measures;
- if only the FDH is broken down to an elementary level then functional model measures only should be recorded, in association with the transaction, user interface and data model measures.

Thus for the purposes of the study a *primitive function* consists of:

1. a single function at the lowest level of a hierarchically decomposed functional model (referred to from now on as a *functional model primitive*) **and/or**
2. a single process and all connected elements at the lowest level of a hierarchically decomposed process model (referred to from now on as a *process model primitive*) **and**
3. the section of the data model upon which the function and/or process acts (referred to from now on as a *data model primitive*).

This approach may be illustrated by a small example taken from the specification of a university department's administration system. In this case, only DFDs have been used to depict the process requirements of the system. At the elementary level there is a process that specifies the production of a class list. Given that Figure 5 depicts the underlying data model for the whole system, Figures 6 and 7 show the relevant process and data model primitives that comprise the single primitive function.

The following five sub-sections fill out the details of the assessment scheme adopted in the study. A number of commercial software specification techniques were briefly described in the previous section - methods for the assessment of functional representations developed using those techniques are therefore now provided. Each of the sections describes the basis for assessment and the actual measures chosen in the scheme. Measures preceded by an asterisk (*) in Tables 1 to 7 are composite measures; that is, they are merely calculated from the values of other measures. The use of composite measures as overall indicators of specification perspective size is supported by Tate and Verner. [1] This approach was therefore extended in the study to consider aspects other than size, for example, interconnection.

In cases where a non-composite system level (macroanalysis) measure is simply the sum of the values obtained for the same primitive function level (microanalysis) measure over all primitive functions in a system it is denoted by placing a 'T' in front of the primitive function measure. Thus the 'TCR' measure in Table 1 is the sum of the values of the 'CR' measure that are obtained for all primitive functions in the system. For example, if a system was made up of four primitive functions and the values of the CR measure for those primitive functions were two, one, three and zero respectively, then the value of the TCR measure for that system would be the sum of these four figures, that is, six.

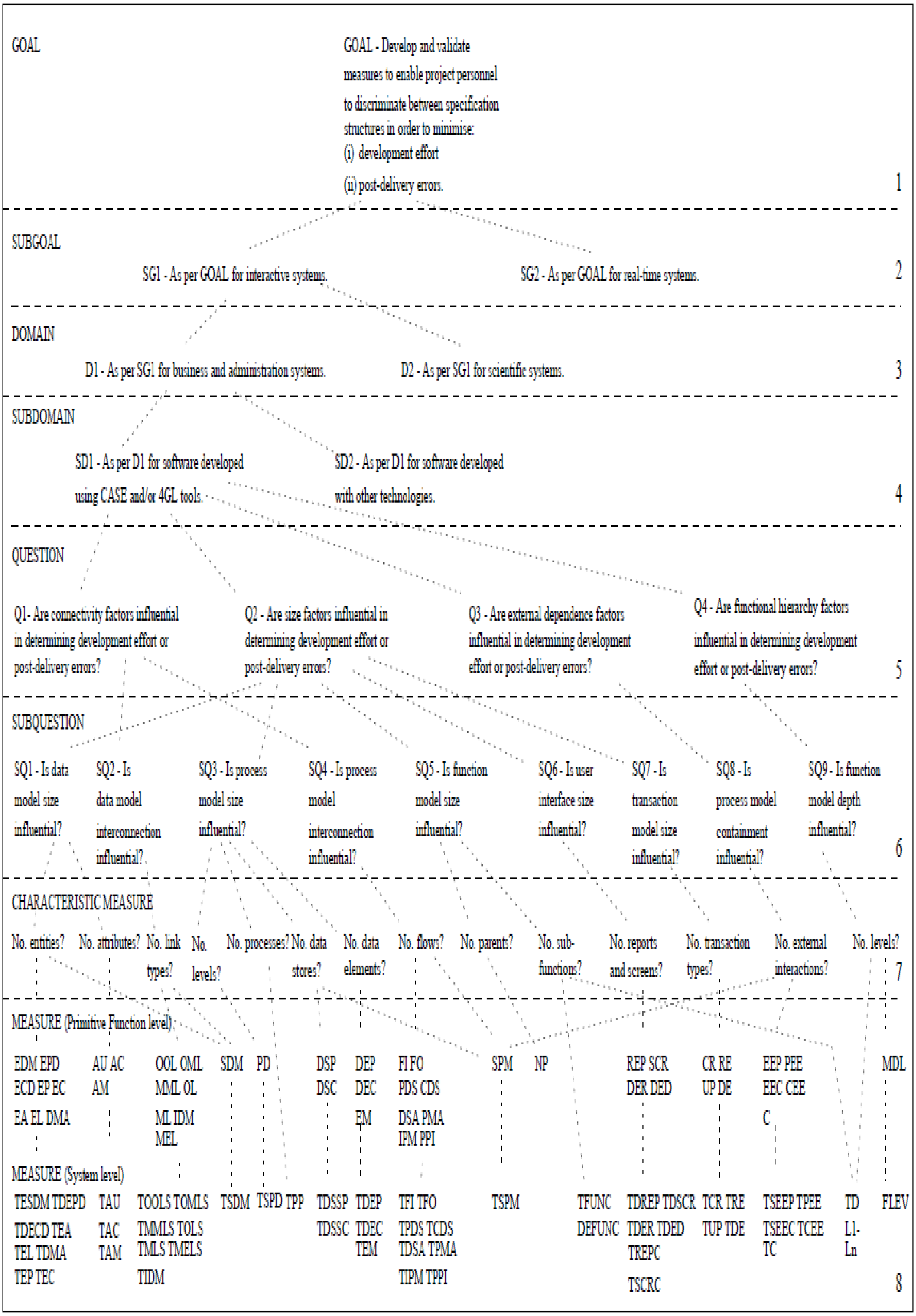

GOAL
GOAL - Develop and validate measures to enable project personnel to discriminate between specification structures in order to minimise:
(i) development effort
(ii) post-delivery errors.
1
SUBGOAL
SG1 - As per GOAL for interactive systems.
SG2 - As per GOAL for real-time systems.
2
DOMAIN
D1 - As per SG1 for business and administration systems.
D2 - As per SG1 for scientific systems.
3
SUBDOMAIN
SD1 - As per D1 for software developed using CASE and/or 4GL tools.
SD2 - As per D1 for software developed with other technologies.
4
QUESTION
Q1- Are connectivity factors influential in determining development effort or post-delivery errors?
Q2 - Are size factors influential in determining development effort or post-delivery errors?
Q3 - Are external dependence factors influential in determining development effort or post-delivery errors?
Q4 - Are functional hierarchy factors influential in determining development effort or post-delivery errors?
5
SUBQUESTION
SQ1 - Is data model size influential?
SQ2 - Is data model interconnection influential?
SQ3 - Is process model size influential?
SQ4 - Is process model interconnection influential?
SQ5 - Is function model size influential?
SQ6 - Is user interface size influential?
SQ7 - Is transaction model size influential?
SQ8 - Is process model containment influential?
SQ9 - Is function model depth influential?
6
CHARACTERISTIC MEASURE
No. entities?
No. attributes?
No. link types?
No. levels?
No. processes?
No. data stores?
No. data elements?
No. flows?
No. parents?
No. sub-functions?
No. reports and screens?
No. transaction types?
No. external interactions?
No. levels?
7
MEASURE (Primitive Function level)
EDM EPD ECD EP EC EA EL DMA
AU AC AM
OOL OML MML OL ML IDM MEL
SDM
PD
DSP DSC
DEP DEC EM
FI FO PDS CDS DSA PMA IPM PPI
SPM
NP
REP SCR DER DED
CR RE UP DE
EEP PEE EEC CEE C
MDL
MEASURE (System level)
TESDM TDEPD TDECD TEA TEL TDMA TEP TEC
TAU TAC TAM
TOOLS TOMLS TMMLS TOLS TMLS TMELS TIDM
TSDM
TSPD TPP
TDSSP TDSSC
TDEP TDEC TEM
TFI TFO TPDS TCDS TDSA TPMA TIPM TPPI
TSPM
TFUNC DEFUNC
TDREP TDSCR TDER TDED TREPC TSCRC
TCR TRE TUP TDE
TSEEP TPEE TSEEC TCEE TC
TD L1-Ln
FLEV
8

Figure 3. Goal/Question/Measure paradigm

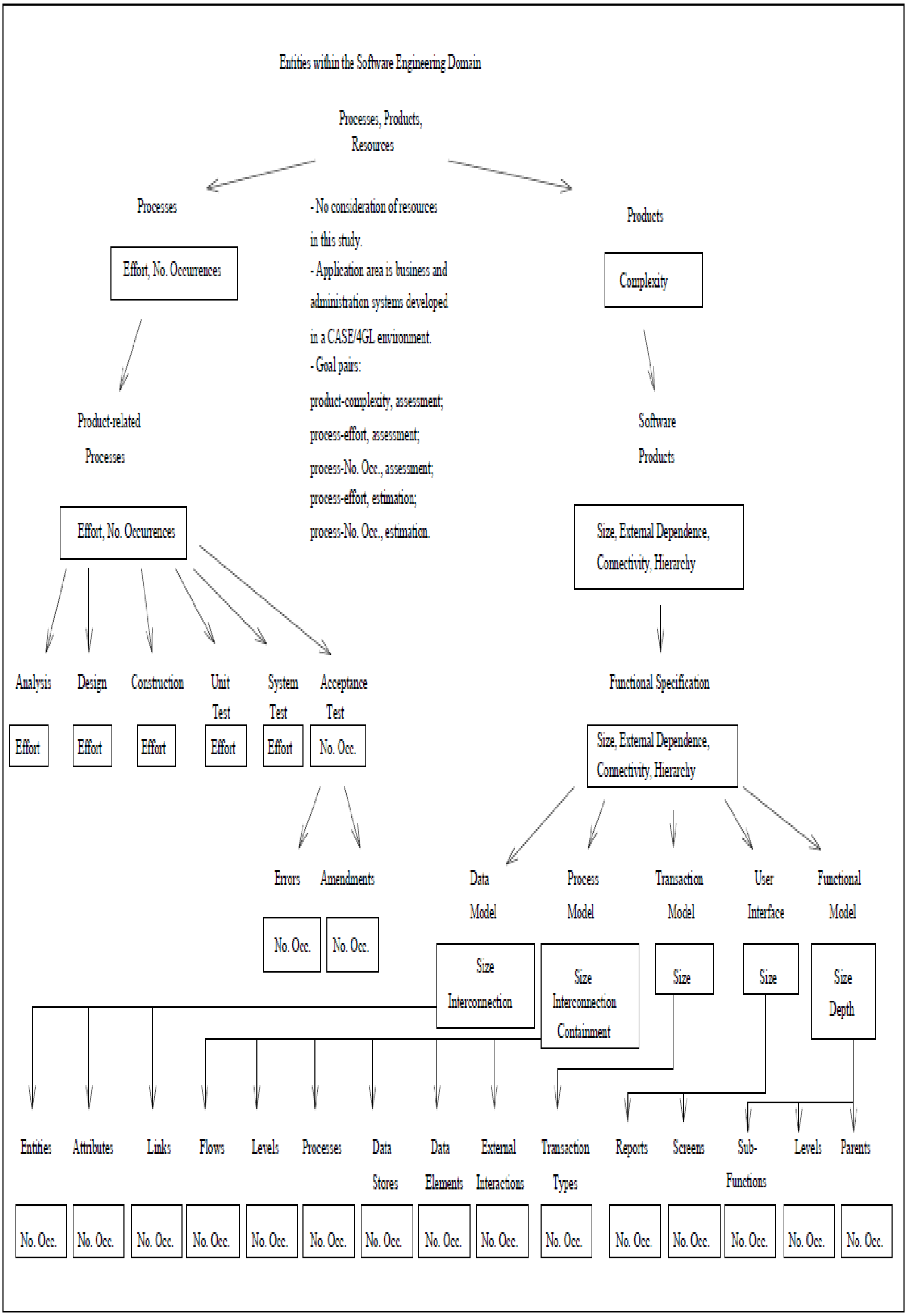

Entities within the Software Engineering Domain
Processes, Products, Resources
Processes
Effort, No. Occurrences
- No consideration of resources in this study.
- Application area is business and administration systems developed in a CASE/4GL environment.
- Goal pairs:
product-complexity, assessment;
process-effort, assessment;
process-No. Occ., assessment;
process-effort, estimation;
process-No. Occ., estimation.
Products
Complexity
Product-related Processes
Effort, No. Occurrences
Software Products
Size, External Dependence, Connectivity, Hierarchy
Analysis
Design
Construction
Unit Test
System Test
Acceptance Test
Effort
No. Occ.
Functional Specification
Size, External Dependence, Connectivity, Hierarchy
Errors
Amendments
Data Model
Process Model
Transaction Model
User Interface
Functional Model
Size Interconnection
Size Interconnection Containment
Size
Size Depth
Entities
Attributes
Links
Flows
Levels
Processes
Data Stores
Data Elements
External Interactions
Transaction Types
Reports
Screens
Sub-Functions
Levels
Parents
No. Occ.

Figure 4. Classification Scheme paradigm

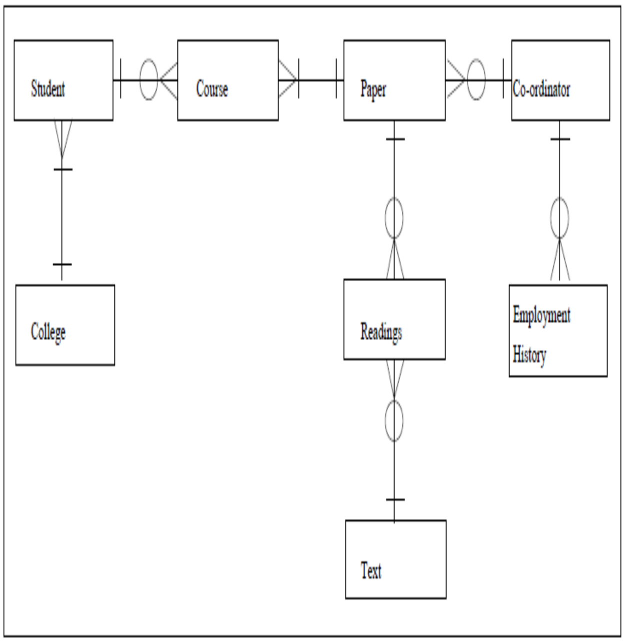


Figure 5. University department system data model example

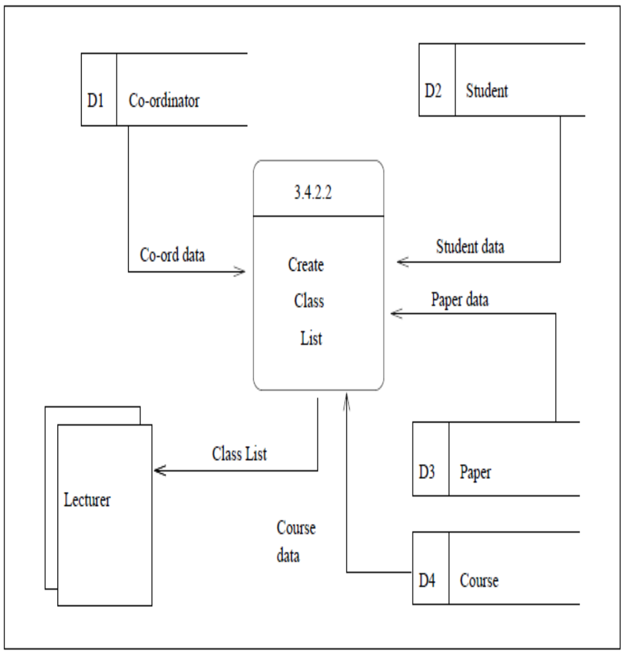


Figure 6. Process model primitive example

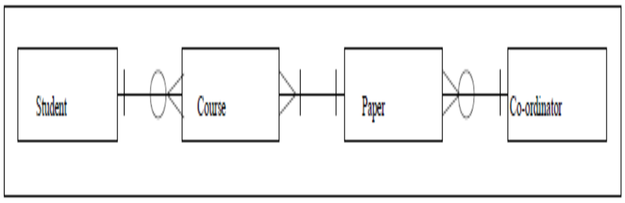


Figure 7. Data model primitive example

3.1. Transaction measures

Elementary transactions of database-oriented systems in the commercial domain always perform one or more of the following operations: create a record (C), read a record or part of a record (R), update a record or part of a record (U) or delete a record (D) (CRUD). Although generally recognised as a data structure-based specification method, due to its basis in entity models, Kerr [7] in fact describes process modelling solely in these terms, in that for each entity in a data model at least three functional modules will be developed, with each module containing the create, update or delete rules for controlling the manipulation of the data. Gray et al. [11] remark that single create, update and delete operations work on only one entity each, but that a read may access several entities in one operation. They therefore suggest that the read operation may be different from the others in terms of complexity. Worsley [22] also found that in a study of enhancement effort for a medium-sized 4GL system, estimates for tasks that required new create transactions were poor, and that the actual re-testing effort required for these enhancements was also generally greater than predicted. Furthermore, the four operations are also weighted differently under a project sizing methodology developed by British Gas. [23]

Four size measures relating to the transaction representation of a specification are therefore included in the analysis scheme, at each of the primitive function and system levels. These are the (T)CR, (T)RE, (T)UP and (T)DE measures, as defined in Table 1. Note that it is the number of operations, not the number of entities operated upon, that is counted in the scheme; the number of entities referenced is assessed in the data model measures. Thus if a single primitive function reads and updates a single entity, both the RE and UP measures for that primitive function should be incremented.

Table 1. Transaction measures

Microanalysis and macroanalysis transaction measures	
(T)CR	(Total) Number of create transactions performed by the primitive function/system
(T)RE	(Total) Number of read transactions performed by the primitive function/system
(T)UP	(Total) Number of update transactions performed by the primitive function/system
(T)DE	(Total) Number of delete transactions performed by the primitive function/system

The transaction measures may be illustrated by the following example. A system comprises three primitive functions, F1, F2 and F3. Function F1 reads data from two entities for subsequent processing by another primitive function; F2 reads data from one entity, displays it on the screen and then updates that entity after input from the user; F3 deletes records from two entities. The transaction measures for these primitive functions therefore take the values shown in Table 2. The system level transaction measures are then easily determined from the primitive function values, using the method described at the end of the previous section. Thus for this same example TCR equals zero, TRE equals three, TUP equals one and TDE equals two.

Table 2. Example primitive function level transaction measures

Measure	F1	F2	F3
CR	0	0	0
RE	2	1	0
UP	0	1	0
DE	0	0	2

3.2. Functional model measures

In cases where the functional hierarchy model is broken down to an elementary level the FDH can form the basis for quantitative analysis of several aspects of a system. Primarily, system size, in terms of the number and distribution of modules, is shown. However the calling structure is also portrayed, through the use of linkages between levels of the hierarchy. Measures of both system size and system depth are therefore available from this representation. These measures are defined in Table 3.

Table 3. Functional model measures

Microanalysis function model measures	
MDL	Maximum decomposition level of the function
NP	Number of parent functions
Macroanalysis function model measures	
DEFUNC	Number of distinct elementary functions in the decomposition
FLEV	Maximum number of function decomposition levels
L1	Number of functions at level 1
L2	Number of functions at level 2
…	…
L*n*	Number of functions at level *n*
*TFUNC	Total functions (L1 + L2 + … + L*n*)
*TD	Total decomposition ((L1 x 1) + (L2 x 2) + … + (L*n* x *n*))

Only two functional model measures are included in the analysis scheme at the primitive function level. The first measure, MDL, is an indication of the level of system depth at which the function is to be implemented. Modules in a hierarchy, and the data elements that they manipulate, are likely to be affected by the processing performed above and below them. Those at higher levels, for example, may be more vulnerable to functional errors, due to the fact that they must control and co-ordinate the processing of often large numbers of modules at lower levels of the hierarchy. Greater caution in development and more extensive testing may therefore be necessary for higher level modules - it is suggested here that the MDL measure may to some degree reflect these requirements. The second measure, NP, quantifies the number of parents that call a single function. By examining the use of common function calls it may be found, for example, that a function that is called or controlled by more than one parent is more difficult to develop than a module with only one parent, given that the plural-parent module may need to cope with different data under different calling conditions.

A completely different set of measures than that used for primitive function level assessment are included in the analysis scheme at the system level. They are mainly indicators of system size, but there is also some consideration of the impact of the decomposition structure on overall complexity. DEFUNC is the count of all the distinct elementary system functions, that is, those functions in the hierarchy that call no other functions. Note that each distinct function is only counted once, even though it may be called in several instances. This is to reflect the fact that the function will only be developed once, even if it may be used more than once. The FLEV measure is similar in principle to the primitive function level MDL measure. If a particular system hierarchy is decomposed down to a maximum of six levels, that is, one or more elementary functions must be traced back through at most six calling modules to reach the highest-level system description, then FLEV will equal six for that system. Comparative indications of system depth may contribute in a relative manner to overall system complexity. The L1 to L*n* measures are derived only for the determination of the TFUNC and TD measures of decomposition. Note that level 1 is one below that at which the highest-level system description is depicted. The TD measure uses relative weightings to provide an overall assessment of the complete decomposition structure. This is in an attempt to assess the relative contributions of both hierarchy depth and breadth to total system complexity.

The hierarchy depicted in Figure 8 provides a basis for illustrating the derivation of the measures just described. If we choose function ‘Store Item’ for microanalysis assessment the MDL measure would take a value of two; this is because function ‘Store Item’ occurs at level 2 in the hierarchy. The NP measure for this same function would equal one as it has just the one parent function, that is, function ‘Receive and Store Item’. If, on the other hand, we were to choose function ‘Read Item and Required’ for assessment, MDL would be assigned the value of five, as this is the highest value level at which it appears. Furthermore NP would equal two for this function, as it is called by both function ‘Read Updated Items’ and function ‘Calculate Deficit’. Any repeat of a function such as in this case is denoted on the diagram by an asterisk (*) after the function name.

Concentration on the complete hierarchy also enables the derivation of the macroanalysis functional model measures. The DEFUNC measure is defined as the number of distinct elementary functions in the decomposition. For this example the distinct elementary functions, that is, those that do not call any other functions, are functions ‘Receive Item’, ‘Store Item’, ‘Read Item and Required’, ‘Request Missing Items’ and ‘Report Requirements’. Hence DEFUNC for this example equals five. The L1 to L*n* measures are as follows: L1 equals two (functions ‘Receive and Store Item’ and ‘Determine Requirements’), L2 equals four (functions ‘Receive Item’, ‘Store Item’, ‘Identify Missing Items’ and ‘Request Missing Items’), L3 equals three (functions ‘Read Updated Items’, ‘Request Missing

Items' and 'Report Requirements'), L4 equals two (functions 'Read Item and Required' and 'Calculate Deficit') and L5 equals one (function 'Read Item and Required'). The FLEV measure is equal to *n* in the L*n* measure, that is, FLEV equals five for this example. The TFUNC and TD measures are then directly computable from the previously derived measurement values.

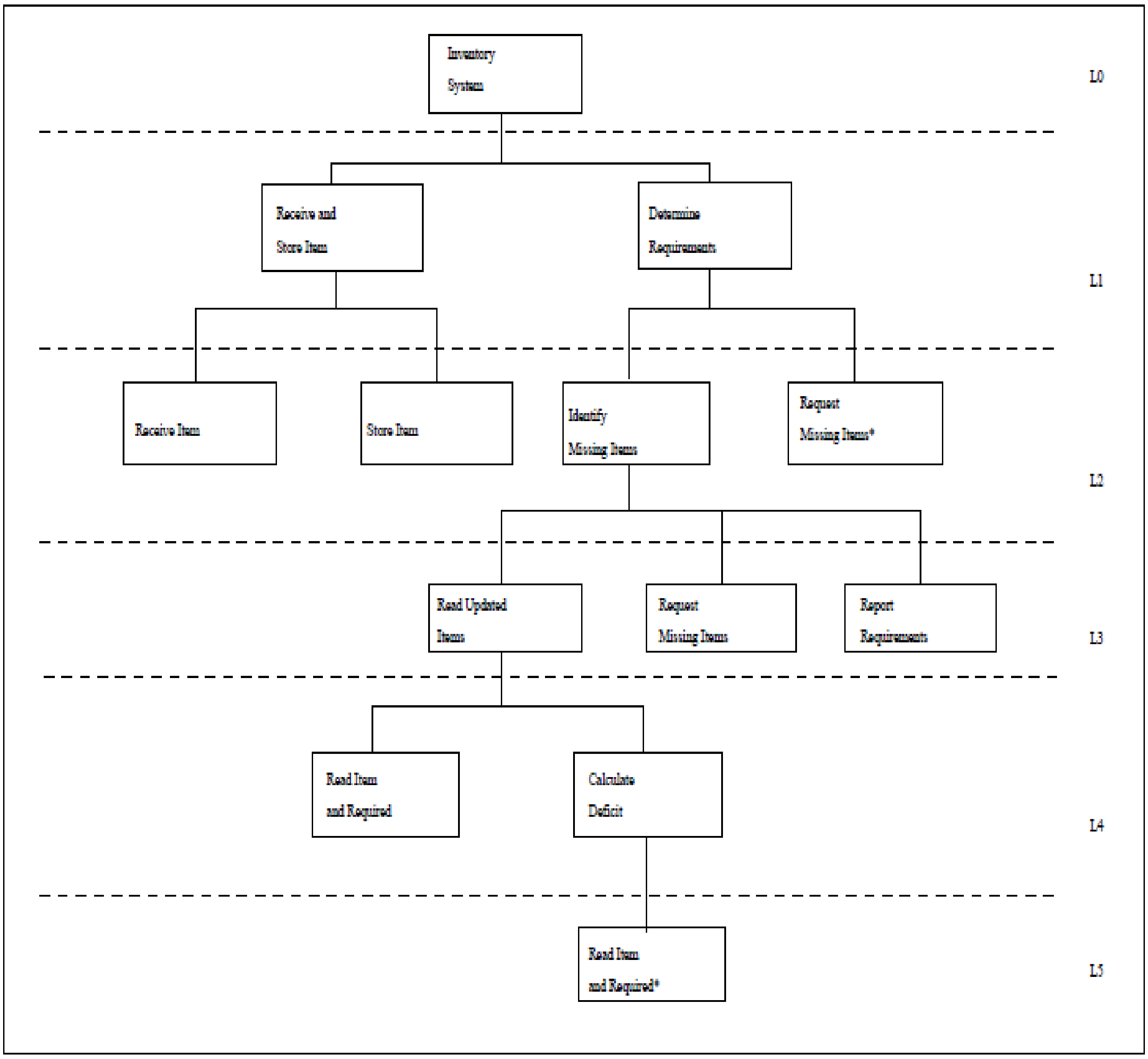


Figure 8. Functional decomposition hierarchy example

3.3. User interface measures

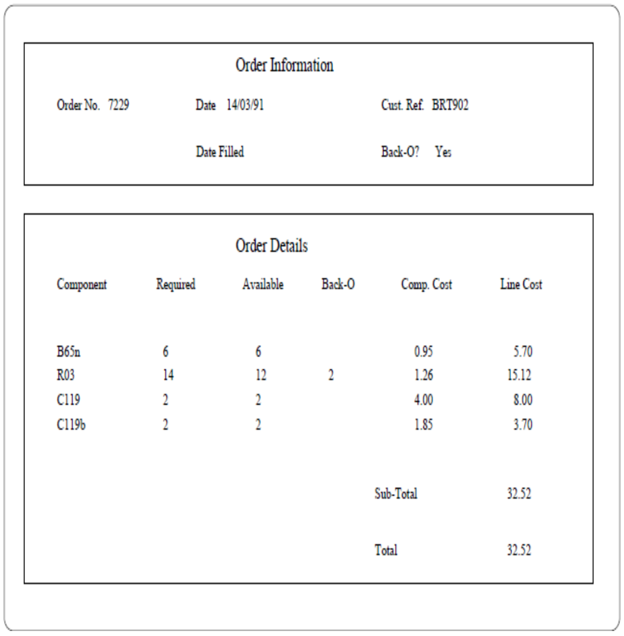
Boehm et al. [24] and Lin [6] suggest that the number of user-required screens and reports in a 4GL-developed system will have a direct effect on development effort, as the construction of approved report and screen layouts can form a significant part of 4GL system development. Worsley, [22] for example, found that the time taken for report development with a 4GL was longer for reports that included complex layouts and that accessed large numbers of tables.

The primitive function level user interface measures are therefore included to reflect findings such as this - the layout of both reports and screens is considered here to be related to the number of data elements that are produced on each. It is also assumed that a primitive function that uses more screens and produces more reports will be more complex than one that uses fewer of these representations. The REP and SCR measures, as defined in Table 4, therefore consider the number of complete reports and screens that are referred to by a primitive function, and the DER and DED measures consider the number of distinct elements used on those reports and screens. A distinct element is an actual data element, not a label, header or footer, that should be counted only once for each report or screen on which it appears, no matter how many times that element may be used on a single report or screen. The system level measures for this representation, which also appear in Table 4, are similar to but not the same as the primitive level measures just described. The TDREP and TDSCR measures are simply the total number of distinct, that is, different, reports and screens that are employed by a system. TDER and TDED are directly comparable to the DER and DED measures discussed above; thus they are simply the sums of the DER and DED values for all primitive functions. The two other indicators, TREPC and TSCRC, equate to the total number of times that reports and screens are used in the system.

Table 4. User interface measures

Microanalysis user interface measures	
REP	Number of reports produced by the primitive function
SCR	Number of screens displayed by the primitive function
Microanalysis and macroanalysis user interface measures	
(T)DER	(Total) Number of distinct elements reported by the primitive function/system
(T)DED	(Total) Number of distinct elements displayed by the primitive function/system
Macroanalysis user interface measures	
TDREP	Total number of distinct reports produced by the system
TREPC	Total number of report calls performed by the system
TDSCR	Total number of distinct screens displayed by the system
TSCRC	Total number of screen calls performed by the system

The sample screen shown in Figure 9 may be associated with a given primitive function. If this is the only screen used by the primitive function, and no reports are produced by the function, then the microanalysis measures would take the following values: REP and DER would both equal zero, SCR would equal one and DED would equal thirteen. The thirteen elements displayed are Order No., Date, Date Filled, Cust. Ref., Back-O?, Component, Required, Available, Back-O, Comp. Cost, Line Cost, Sub-Total and Total.


Order Information

Order No. 7229 Date 14/03/91 Cust. Ref. BRT902

Date Filled Back-O? Yes

Order Details

Component	Required	Available	Back-O	Comp. Cost	Line Cost
B65n	6	6		0.95	5.70
R03	14	12	2	1.26	15.12
C119	2	2		4.00	8.00
C119b	2	2		1.85	3.70
				Sub-Total	32.52
				Total	32.52

Figure 9. Screen example

3.4. Process model measures

One of the main benefits of utilising data flow diagrams in commercial software development is the reduction in system complexity that the partitioning process provides. [25] A consideration of the functional complexity associated with this representation should therefore be applicable here. Generally the measures of this model from both analysis levels reflect the assumption that a large process model, in terms of decomposition levels, processes, related data stores and individual data element usage, will result in a proportionally large coded function. Thus the measures from Table 5 that relate to the numbers of levels, processes, stores and elements, that is, TPP, PD, TSPD, DSP, DSC, TDSSP, TDSSC, (T)DEP and (T)DEC, are all indications of process model size. The TPP measure is clearly only applicable to the system-level analysis procedure, given that the microanalysis technique considers only one primitive process at a time. Therefore TPP is simply the total number of primitive (lowest-level) processes in the system. In this respect it is very similar to the DEFUNC functional model measure. In the same way, the process model PD and TSPD measures are similar to the functional model MDL and FLEV measures. They are included to assess the depth of processing that is to be implemented in the final system.

It is acknowledged that the data store measures DSP, DSC, TDSSP and TDSSC may not be of significance, as the number of stores may be determined by an arbitrary decision of the analyst. One analyst may prefer to have a distinct data store for each entity, whereas another may group entity-views into data stores simply for representational convenience. For purposes of completeness, however, these measures will still be taken for each system. The use of the words ‘non-file’ in the (T)DEP and (T)DEC definitions reflects the fact that all file-related elements are assessed in the evaluation of the data model with the (T)AU and (T)AC measures. There could be considerable overlap in these two pairs of measures; the ‘non-file’ condition, however, enables the sole consideration of data elements other than those stored by the system that are (i) input by the user or by other external systems/processes and (ii) produced on the screen and in report formats.

Process model interconnection should reflect the degree of coupling that will be implemented in the final system. [26] Interconnection at this level may be related to Henry and Kafura’s design phase Information Flow measure, [14] in that the fan-in and fan-out measures may be approximated by the flows-in and flows-out measures of this proposal. Since it is generally accepted that DFD process interconnection should be minimised to lessen complexity, [1,27] lower values of interconnection measures such as (T)FI and (T)FO should have a positive impact on the ease of development.

Table 5. Process model measures

Microanalysis process model measures	
DSP	Number of distinct data stores providing data
DSC	Number of distinct data stores consuming data
PD	Process depth (Number of parent processes up to level 1)
EEP	Number of distinct external entities providing data
EEC	Number of distinct external entities consuming data
*SPM	Size (Process model) (PMA + DSP + DSC + EEP + EEC)
Microanalysis and macroanalysis process model measures	
(T)FI	(Total) Number of flows into the process/system processes
(T)FO	(Total) Number of flows out of the process/system processes
(T)PDS	(Total) Number of provisions from data stores
(T)CDS	(Total) Number of consumptions by data stores
(T)DEP	(Total) Number of non-file elements produced by the process/system processes
(T)DEC	(Total) Number of non-file elements consumed by the process/system processes
(T)PPI	(Total) Number of process-to-process flows into the process/system processes
(T)PEE	(Total) Number of provisions from external entities
(T)CEE	(Total) Number of consumptions by external entities
*(T)PMA	(Total) Process model access ((T)FI + (T)FO)
*(T)IPM	(Total) Interconnection (Process model) ((T)FI x (T)FO)^2
*(T)DSA	(Total) Data store access ((T)PDS + (T)CDS)
*(T)EM	(Total) Element manipulation ((T)DEP + (T)DEC))
*(T)C	(Total) Containment ((T)PEE + (T)CEE)
Macroanalysis process model measures	
TPP	Total number of primitive processes
TDSSP	Total number of distinct system data stores providing data
TDSSC	Total number of distinct system data stores consuming data
TSPD	Total system process depth (Number of process levels)
TSEEP	Total number of distinct system external entities providing data
TSEEC	Total number of distinct system external entities consuming data
*TSPM	Total size (Process model) (TPMA + TDSSP + TDSSC + TSEEP + TSEEC)

The level of system containment is also considered to be important. Dependence on data supplied from outside the system boundaries may have an effect on the ease of system implementation and maintenance, especially where control over the form and validity of the data is out of the developer's hands. External entities that receive data, on the other hand, often require this information in some form of report. For example, a packing slip may be sent to a warehouse, or an invoice to a customer. Therefore development and maintenance of the relevant process would also involve the creation or consideration of a report form and the incorporation of extra processing to produce that report. The EEP, EEC, TSEEP, TSEEC, (T)PEE and (T)CEE measures are therefore included to reflect the impact of process model containment. The measures differ in the fact that the first four simply consider the number of external entities that are involved in the operation of a primitive function or a system, whereas the final four assess the actual number of interactions between system processes and external entities. This follows the same approach as that used in the collection of the DSP, DSC, TDSSP, TDSSC, (T)PDS and (T)CDS measures.

The process model primitive example shown earlier in the section (Figure 6) may be useful in illustrating the derivation of microanalysis process model measures. If we know from the user interface assessment that the Class List report contains fourteen data elements then all of the non-composite process model measures can be determined immediately, as shown in Table 6.

Table 6. Example process model primitive measures

Measure	Value
FI	4
FO	1
DSP	4
PDS	4
DSC	0
CDS	0
DEP	14
DEC	0
PPI	0
PD	3
EEP	0
PEE	0
EEC	1
CEE	1

3.5. Data model measures

The final set of specification product measures considers aspects of size and interconnection associated with the data model representation. The size of a data model will provide a first-cut, basic indication of the amount of processing that is to be performed on it; that is, a larger data model implies a greater degree of processing to reference, manipulate and/or write to the entities and individual attributes involved. Data model size may also be influential in the estimation of maintenance tasks, particularly for data retrieval systems, where the structure of the existing data will have an impact on how new data should be incorporated into the system. [11,28] Measures from this class in the current analysis scheme, as defined in Table 7, include EDM, TESDM, EPD, TDEPD, ECD, TDECD, (T)EP, (T)EC, (T)AU, (T)AC and (T)EL.

The number of entities involved in system operations are considered by the EDM, TESDM, EPD, TDEPD, ECD, TDECD, (T)EP and (T)EC measures. The first two measures are quite straightforward - they are simply counts of the number of entities that are referenced or traversed by a primitive function or by a system. The other measures then consider the types of references that the entities undergo, whether providing data, consuming data or both, as a result of system operations. Use of the measures is based on the same approach as that used for the process model assessment. That is, EPD through to TDECD consider the number of distinct entities referenced, whereas (T)EP and (T)EC consider the actual number of references.

Table 7. Data model measures

Microanalysis data model measures	
EDM	Number of entities in the data model primitive
EPD	Number of distinct entities providing primitive function data
ECD	Number of distinct entities consuming primitive function data
OOL	Number of 1:1 links between entities in the data model primitive
OML	Number of 1:*n* links between entities in the data model primitive
MML	Number of *n*:*m* links between entities in the data model primitive
OL	Number of optional links in the data model primitive
ML	Number of mandatory links in the data model primitive
MEL	Number of exclusive links between entities in the data model primitive
*IDM	Interconnection (Data model) (OOL + OML + MML)
*SDM	Size (Data model) (EDM + IDM)
Microanalysis and macroanalysis data model measures	
(T)EP	(Total) Number of entity provisions
(T)EC	(Total) Number of entity consumptions
(T)AU	(Total) Number of attributes updated by the primitive function/system
(T)AC	(Total) Number of attributes consumed by the primitive function/system
(T)EL	(Total) Number of entity look-ups performed by the primitive function/system
*(T)EA	(Total) Entity access ((T)EP + (T)EC)
*(T)AM	(Total) Attribute manipulation ((T)AU + (T)AC)
*(T)DMA	(Total) Data model access ((T)EA + (T)EL)
Macroanalysis data model measures	
TESDM	Total number of entities in the system data model
TDEPD	Total number of distinct entities providing data
TDECD	Total number of distinct entities consuming data
TOOLS	Total number of 1:1 links between entities in the system data model
TOMLS	Total number of 1:*n* links between entities in the system data model
TMMLS	Total number of *n*:*m* links between entities in the system data model
TOLS	Total number of optional links between entities in the system data model
TMLS	Total number of mandatory links between entities in the system data model
TMELS	Total number of exclusive links between entities in the system data model
*TIDM	Total interconnection (Data model) (TOOLS + TOMLS + TMMLS)
*TSDM	Total size (Data model) (TESDM + TIDM)

Gray et al. [11] suggest that data model measures should also include some consideration of the amount of data actually passed to and from the database. They therefore propose that this could be derived from the number of attributes flowing in the system - for create and delete operations this would be the number of attributes in the entity referenced; for the update and read operations it would be the number of actual attributes referenced. Although this suggestion was developed independently of the current study, these measures equate precisely to the attributes-updated and attributes-consumed measures ((T)AU and (T)AC) of the current proposal. The entity look-up count, (T)EL, should be incremented only when an entity is referenced purely for validation purposes, that is, when an entity is read only to ensure that a particular field value is allowed - the entity's data is not actually used in the process. An entity may, however, be counted more than once for a given system/primitive function if it is accessed both to supply data for processing and for look-up validation.

The interconnection among entities using various relationship types also gives an indication of processing requirements. For example, a one-to-many relationship suggests a hierarchical link, such as that for orders and order lines, providing an insight as to how the relevant data will be entered and processed. Eglington [29] suggests, in fact, that the complexity of many data processing systems is mainly contained in the relationships between records. It would therefore seem worthwhile to consider entity relationship link types at the logical level in an assessment of complexity. Participation requirements may also be important. A mandatory connection, for example, may indicate a need for validation during processing to ensure that no null entries are supplied. Bushell [30] and Keuffel [31] both state that many-to-many (*n:m*) relationships are difficult to implement. Bushell [30] also suggests that connections between entities should be minimised because, if there are several ways of traversing system data for essentially the same purpose, different paths will be used in different cases. Moreover, subsequent changes will be made more difficult. The existence of only one path therefore ensures a standard approach. To this end, the interconnection measures OOL through to TMELS are also included in the analysis scheme.

Derivation of the OOL, OML, MML, TOOLS, TOMLS and TMMLS counts may not be obvious when it comes to the assessment of certain relationship types. Recommendations are therefore made for the following situations:

- a. recursive relationships - as these have no direct impact on the difficulty of development, they should be assessed in the same way as any other relationship
- b. multiple relationships -
    1. where more than one distinct relationship exists between two entities on a given system or primitive data model, and where the system/primitive function being assessed may traverse any of these relationships at one time, each relationship should be counted separately as part of the assessment;

2. where more than one distinct relationship exists between two entities on a given system or primitive data model, and where the system/primitive function being assessed may traverse only a subset of those relationships at one time, each relationship in the subset should be counted as part of the assessment.

Returning to the primitive function depicted in Figures 6 and 7, the microanalysis data model measures can now also be determined. For simplicity's sake let us state that each of the entities in the data model primitive contains five attributes and that none of the entities are referenced for look-up purposes. The non-composite data model primitive measures are therefore assigned the values shown in Table 8.

Table 8. Example data model primitive measures

Measure	Value
EDM	4
EPD	4
EP	0
ECD	0
EC	0
AU	0
AC	20
EL	0
OOL	0
OML	3
MML	0
OL	2
ML	4
MEL	0

3.6. Rationale

DeMarco [2] suggests that the effort required to implement a system increases monotonically with increasing specification size, assuming that there is no redundancy in the specification. As CASE helps to ensure that redundancy is kept to a minimum, size measures derived from specification models should prove to be useful in effort determination. DeMarco [2] also claims that the size of a specification model approaches invariance with respect to the decisions of the individual modeller. This would suggest that, for a given system requirement, roughly the same functional measurement values will be obtained irrespective of the modeller, particularly when CASE technology is employed. [5,32] This does not imply that the same specification will be produced, but only that the scope and size of the specifications will be similar. However this in itself is an aid to achieving more consistent assessment and analysis. Thus the basis of the complexity analysis scheme in specification representations would still appear to be sound.

A similar comment to that of DeMarco's above is made by Rudolph [33] in a discussion on functional size assessment - the remark is equally applicable to any type of functional analysis, however. Rudolph suggests that by its very nature functional assessment should not in the first instance reflect 'external' factors such as programmer and organisational experience or the effects of various implementation approaches. Rather, the absolute value of the measured attribute should be established first, and then adjusted if required. Moreover, Chen and Norman [34] suggest that the use of a graphic interface in many of the tools enables easier learning and use, lessening the impact of tool experience on effort requirements. All of these factors lend support to the approach adopted in this study, that is, the assessment of functional complexity with no initial adjustment for other factors.

Further support for the purely functional approach is indirectly provided by the advantage of application portability that many CASE tools now provide. [35] Tate and Verner [1] suggest that as projects move through the phases of development more specific methods and techniques are employed. They go on to suggest that this leads to increasing dependence of software products on the target technology, even in a CASE environment. The implication is that the development and implementation methods chosen have an increasing impact on a system's size and that this will have a corresponding impact on development effort. However several CASE tools enable development to be performed irrespective of the eventual implementation platform, as the tools include facilities for implementation on a number of different platforms. [35,36] Thus functional measures should provide a sound basis for the determination and classification of effort requirements in an automated environment up until the implementation phase of development.

In the past, adjustment of functional measures has generally been performed to enable the consideration of special system requirements. It has recently been suggested, however, that systems developed with 4GLs and CASE tools will not require adjustment as the 'special' requirements will be developed as standard: [5,28,34]

> *In the longer term, the ultimate computer-aided systems engineering (CASE) tool will provide all these technical features automatically; we shall only have to think about the information-processing requirements of the problem. In this ultimate situation, the coefficient [of the adjustment factor] will fall to zero, and there will be no further need for a Technical Complexity Adjustment. [28 p. 29]*

Although it is questionable as to whether we have reached the age of the 'ultimate' CASE tool, this assumption is the basis for the current proposal, that is, that useful assessment can be performed based solely on systems' functional requirements. As 4GLs and CASE tools tend to specify functional transactions rather than procedural components, Verner et al. [37] suggest that a specification produced with these tools will be a closer representation of the pure inherent function required. Moreover, since development using these tools is often based around a central repository, the likelihood of duplicated and inconsistent work among development teams should be reduced, providing a basis for more consistent and accurate assessment of functional characteristics. It is certainly possible that measures from a specification will be related to the functional value of the system, based on the assertion that transactions comprised of larger and more complex specifications provide more function to the user. [37] Appropriate measures of functional size and complexity are therefore required. Those proposed here are a first response to this need.

A major criticism of many existing assessment methods is their consideration of only one aspect of complexity, for example, control-flow, size or data flow. [38,39] The proposed analysis scheme should at least partially overcome this problem, in that size, interconnection, data flow, data structure, process coupling and overall function are all assessed in some way. The comprehensive approach that has been adopted in this proposal is supported by Tate and Verner [5] and Wrigley and Dexter; [40] they suggest that appropriate specification measures should come from data structure and data flow models and from aspects of the proposed user interface. Each of these representations has been considered in the overall analysis scheme. The general expectation underlying this approach is that systems or primitive functions that return high values for the transaction, functional model, user interface, process model and/or data model measures will be more time-consuming and error-prone to develop than systems/primitive functions that result in low measures. Due to its more comprehensive approach, the scheme also includes measures that are applicable to software specification products not normally considered. Grady's examination of software development work-product analysis states that the design stage ''...work products for prototypes and data dictionaries because they are widely used today, although they represent two cases where metrics research has been very limited (and so no metrics are shown)''. [41 p. 30] In the proposed scheme the data dictionary may be partially assessed by the data model and process model measures, and the prototype by the user interface and functional model measures. [42]

The two-level approach to assessment is another important aspect of the overall analysis scheme. Although system-wide measures and indicators are useful, it has been acknowledged that lower-level analysis is necessary for effective project management. Verner et al. [37] and Stevens [43] remark that elementary function-based assessment is required so that resource allocation in subsequent development phases can be carried out more effectively. Moreover, in terms of accuracy, lower-level assessment receives further support - Stevens [43] and IE [44] suggest that a more detailed functional breakdown will provide more accurate measures, due to the reduced variance achieved.

Furthermore, the measures in the scheme are automatically derivable at a very early stage in the development process, increasing the scope for objective and effective estimation and discrimination. According to Grady [41] and Gray et al., [11] one of the most promising aspects of CASE is the facility for automatic, 'on-line' delivery of measures and estimates to the project manager. The notion of automatic data collection and analysis has widespread support and would appear to be essential in an unobtrusive form if assessment data analysis is to become a useful, integral part of the development process. [45,46] Automatic collection would also enable more effective progress reports to be produced - as long as subsequent functional changes to a system were incorporated into the current specification models, revised measures and/or estimates could be automatically generated for the project manager during all subsequent stages of development.

4. SUMMARY

This paper has described the structured development of a comprehensive functional complexity assessment scheme that is based on the specifications of commercial software systems. The impact of development automation was examined, resulting in the suggestion that software specification notations would prove to be useful as models for analysis. Various specification representations were therefore considered, and measures appropriate to the complexity assessment task were determined for each perspective using the GQM and Classification Scheme paradigms. Small examples were also provided to illustrate the derivation of the various measurement values. This was followed by a discussion of the rationale upon which the measurement selection had been based.

Small-scale validation of the scheme in the estimation of development effort has already been undertaken, with considerable success. [20] It is envisaged that continued use of the assessment scheme in other automated projects will produce significantly larger data sets, which in turn should lead to greater accuracy in the classification and estimation tasks. This does, however, depend on the willingness of CASE product users to participate in studies of this type. Presuming that this will be forthcoming, it is likely that the scheme will be iteratively refined until only the most influential measures are collected. It is hoped that this will then lead to the incorporation of assessment facilities in actual development tools. This will enable more objective, non-intrusive, less error-prone collection of the data to be carried out without the need for time-consuming manual collection. It will also mean that analysis and prediction may be performed in the background of development as an integral part of a project. Tate [47] and Tate and Verner [1] also suggest that on-workbench data, relating to development effort, will soon be collected automatically within CASE environments. Collection of project management data will therefore also be more precise and cost-effective. All of these factors will encourage continuing refinement of any equations developed, providing relevant feedback to managers whenever required. Only under these circumstances is it likely that functional assessment will become more readily perceived as a necessary and worthwhile task within software development.

ACKNOWLEDGEMENTS

The author would like to thank the referees for their constructive comments. He is also grateful for the financial support provided for this study by the Cambridge Commonwealth Trust, the New Zealand Vice-Chancellors Committee, British Telecom plc, Clare College, Cambridge, the Cambridge University Engineering Department and the University of Otago. The work described in this paper was carried out while the author was a graduate student at Cambridge University.

REFERENCES

1 Tate, G and Verner, J ‘Approaches to Measuring Size of Application Products with CASE Tools’ Information and Software Technology Vol 33 No 9 (November 1991) pp 622-628.

2 DeMarco, T Controlling Software Projects Yourdon New York (1982).

3 DeMarco, T ‘An Algorithm For Sizing Software Products’ ACM SIGMetrics Performance Evaluation Review Vol 12 No 2 (1984) pp 13-22.

4 Ramamoorthy, C V, Prakash, A, Tsai, W -T and Usuda, Y ‘Software Engineering: Problems and Perspectives’ Computer (October 1984) pp 191-209.

5 Tate, G and Verner, J ‘Software Metrics for CASE Development’ in Proceedings COMPSAC ‘91 Tokyo (1991) pp 565-570.

6 Lin, C -Y ‘Systems Development With Application Generators: An End User Perspective’ Journal of Systems Management (April 1990) pp 32-36.

7 Kerr, J M ‘The Information Engineering Paradigm’ Journal of Systems Management (April 1991) pp 28-35.

8 Nelson, M S ‘Computer Aided Software Engineering (CASE)’ Paper 645 Master of Business Administration University of Otago, Dunedin (February 1990).

9 Kilov, H ‘Conventional and Convenient in Entity-Relationship Modeling’ ACM SIGSoft Software Engineering Notes Vol 16 No 2 (April 1991) pp 31-32.

10 Karimi, J and Konsynski, B R ‘An Automated Software Design Assistant’ IEEE Transactions on Software Engineering Vol 14 No 2 (February 1988) pp 194-210.

11 Gray, R H M, Carey, B N, McGlynn, N A and Pengelly, A D ‘Design Metrics for Database Systems’ BT Technology Journal Vol 9 No 4 (October 1991) pp 69-79.

12 Benwell, G L, Firns, P G and Sallis, P J ‘Deriving Semantic Data Models from Structured Process Descriptions of Reality’ Journal of Information Technology Vol 6 No 1 (March 1991) pp 15-25.

13 Paulson, D and Wand, Y ‘An Automated Approach to Information Systems Decomposition’ IEEE Transactions on Software Engineering Vol 18 No 3 (March 1992) pp 174-189.

14 Henry, S and Kafura, D ‘Software Structure Metrics Based on Information Flow’ IEEE Transactions on Software Engineering Vol 7 No 5 (September 1981) pp 510-518.

15 Chapin, N ‘A Measure of Software Complexity’ in Proceedings 1979 National Computer Conference New York (1979) pp 995-1002.

16 Basili, V R and Rombach, H D ‘The TAME Project: Towards Improvement-Oriented Software Environments’ IEEE Transactions on Software Engineering Vol 14 No 6 (June 1988) pp 758-773.

17 Basili, V R and Weiss, D M ‘A Methodology for Collecting Valid Software Engineering Data’ IEEE Transactions on Software Engineering Vol 10 No 6 (November 1984) pp 728-738.

18 Shepperd, M ‘Design Metrics: An Empirical Analysis’ Software Engineering Journal (January 1990) pp 3-10.

19 Bush, M E and Fenton, N E ‘Software Measurement: A Conceptual Framework’ Journal of Systems and Software Vol 12 (1990) pp 223-231.

20 MacDonell, S G ‘Quantitative Functional Complexity Analysis of Commercial Software Systems’ PhD Dissertation, University of Cambridge, Cambridge (1992).

21 MacDonell, S G ‘Overcoming Theoretical Objections to Software Complexity Measurement’ Forthcoming.

22 Worsley, L M ‘Project Management and the Use of Metrics in a Fourth Generation Environment’ in Proceedings 8th European Oracle Users Group Conference Madrid (April 1990).

23 British Gas Bang Metric Analysis Document Num. 000763 Process Support British Gas plc, Dorking (June 1991).

24 Boehm, B W, Gray, T E and Seewaldt, T ‘Prototyping Versus Specifying: A Multiproject Experiment’ IEEE Transactions on Software Engineering Vol 10 No 3 (May 1984) pp 290-302.

25 Keuffel, W ‘The Structured Specification Data Dictionary’ Computer Language (USA) (July 1991) pp 29-34.

26 Tsai, J J -P and Ridge, J C ‘Intelligent Support for Specifications Transformation’ IEEE Software (November 1988) pp 28-35.

27 Tan, K P, Chua, T S and Lee, P -T ‘AUTO-DFD: An Intelligent Data Flow Processor’ The Computer Journal Vol 32 No 3 (1989) pp 194-201.

28 Symons, C R Software Sizing and Estimating: Mk II FPA (Function Point Analysis) John Wiley & Sons Chichester (1991).

29 Eglington, D ‘Cost-Effective Computer System Implementation in Medium Sized Companies’ in Gillies, A (ed) Case Studies in Software Engineering Salford University Business Services Salford (March 1991) pp 56-59.

30 Bushell, C J ‘The Strengths Of Data Modelling’ in Proceedings 18th CAE Computer Conference Australia (1987) pp 105-117.

31 Keuffel, W ‘Exploring ERD Tools’ Computer Language (USA) (April 1991) pp 27-35.

32 Robinson, K ‘Putting the SE into CASE’ in Spurr, K and Layzell, P (eds) CASE: Current Practice, Future Prospects John Wiley & Sons Chichester (1992) pp 1-20.

33 Rudolph, E E Measuring Information Systems Seminar Guide and Additional Notes Auckland (1987).

34 Chen, M and Norman, R J ‘A Framework for Integrated CASE’ IEEE Software (March 1992) pp 18-22.

35 Banker, R D and Kauffman, R J ‘Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study’ MIS Quarterly (September 1991) pp 375-401.

36 Brown, D W, Carson, C D, Montgomery, W A and Zislis, P M ‘Software Specification and Prototyping Technologies’ AT\&T Technical Journal (July/August 1988) pp 33-45.

37 Verner, J, Tate, G, Jackson, B and Hayward, R G ‘Technology Dependence in Function Point Analysis: A Case Study and Critical Review’ in Proceedings 11th International Conference on Software Engineering Pittsburgh PA (1989) pp 375-382.

38 Weyuker, E J ‘Evaluating Software Complexity Measures’ IEEE Transactions on Software Engineering Vol 14 No 9 (September 1988) pp 1357-1365.

39 Longworth, H D, Ottenstein, L M and Smith, M R ‘The Relationship Between Program Complexity And Slice Complexity During Debugging Tasks’ in Proceedings COMPSAC ‘86 Chicago IL (1986) pp 383-389.

40 Wrigley, C D and Dexter, A S ‘A Model for Measuring Information System Size’ MIS Quarterly (June 1991) pp 245-257.

41 Grady, R B ‘Work-Product Analysis: The Philosopher’s Stone of Software?’ IEEE Software (March 1990) pp 26-34.

42 Clarke, R ‘A Contingency Approach to the Application Software Generations’ ACM SIGBIT Data Base (Summer 1991) pp 23-34.

43 Stevens, O B Project Sizing Methodology Development Assurance Services Report - Lloyd’s Bank London (August 1990).

44 IE IE-Metrics Knowledge Base James Martin & Company Reston VA (November 1989).

45 Henry, S and Lewis, J ‘Integrating Metrics into a Large-Scale Software Development Environment’ Journal of Systems and Software Vol 13 (1990) pp 89-95.

46 Norman, R J and Chen, M ‘Working Together to Integrate CASE (Guest Editors’ Introduction)’ IEEE Software (March 1992) pp 13-16.

47 Tate, G ‘Management, CASE and the Software Process’ in Proceedings 12th New Zealand Computer Conference Dunedin (1991) pp 247-256.