Responsibility sharing between sophisticated users and professionals in structured prototyping

N Pliskin and P Shoval

A scale of data-processing sophistication levels exists among users of information technology. At one end of the scale are users whose interaction with computers is minor and nondevelopmental in nature. At the other end are users who actually develop small applications.

A natural result of the increase in the data-processing sophistication of users is end-user prototyping, which can either be evolutionary, leading to an operational user-developed application, or lead to a throw-away prototype, where development of the operational production system is done by professionals. Drawbacks associated with the unprofessional data-processing conduct of end-users and the lack of structure inherent in prototyping triggered this work on structured methodological support for end-user prototyping. The basis of the approach is structured prototyping, a development framework for information-systems professionals. Depending on the level of user sophistication and on system complexity, it is proposed to transfer selectively structured prototyping tasks from professionals to users. A case study is presented to demonstrate a scenario of responsibility sharing between a sophisticated user and a professional.

system development, prototyping, structured prototyping, structured methodologies, user sophistication, responsibility sharing

Throughout most of the short history of data processing (DP), system development has involved two parties, professionals and end-users. Traditionally, in terms of DP professionalism, these two groups did not overlap: end-users were naive and devoid of DP skills, while professionals were trained for and had gained experience in DP[1]. Technological progress has, however, significantly altered the end-user population[2,3]. First, the proportion of computer users in the general population increased. Second, a new class emerged, composed of non-naive users who have acquired DP sophistication through academic or professional training, from practice as microcomputer users, or as subscribers to information centres[4,5].

Several end-user typologies published in recent years[6–9] suggest that a scale of sophistication levels is inherent in the population of sophisticated users. This study highlights the two ends of the sophistication scale. At the lower end is the least sophisticated user, termed hereafter MinUser. Unlike naive end-users, who remain rather passive on interacting with computers, MinUsers are active creators of limited-scale applications in support of professional tasks. For example, they might interactively apply a spreadsheet package for financial analysis or adopt a database management package for interactive creation and maintainance of data files. The resulting familiarity with input, output, and menu interfaces renders MinUsers more capable (than naive users) of specifying interface requirements to system developers.

At the higher end of the scale is the most sophisticated end-user, termed hereafter MaxUser. MaxUsers, termed elsewhere SuperUsers[4], engage in programming (e.g., spreadsheet macros) of advanced user-developed applications (UDAs)[10], actively interact with professionals throughout system development[8], share UDAs with other end-users, and informally serve as liaisons between users and professionals.

UDAs are highly regarded as a valuable resource in the reduction of application backlogs and software-maintenance loads[7]. At the same time, UDAs are often criticized for deficiencies in efficiency, integrity, documentation, backup, and security[9–12]. Are these deficiencies unavoidable? Some can be overcome through

Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, P.O. Box 653, Beer-Sheva 84105, Israel. Professor Pliskin is currently at Dept of Computer Information Systems, Suffolk University, Beacon Hill, 8 Ashburton Place, Boston, MA 02108, USA

 0950-5849/89/080438-11 $03.00

awareness, training, and proper management[13]. Some UDA deficiencies cannot be overcome by these means alone as end-users, in effect, engage in prototyping when they develop UDAs[1].

Prototyping, the quick development of working models to test out system requirements[14–22], is an approach used by professionals as well. After years of debate, comparing prototyping with structured system development, there is already a reasonable consensus that application of both has value. Generally, it is agreed that prototyping is advantageous in user satisfaction and length of development time, but that in terms of other dimensions its value depends on system complexity[1]. For complex and large systems, for example, development costs are not necessarily reduced because the prototype is usually a 'throw-away' (as opposed to an 'add-on') and is eventually discarded[16]. For nontrivial systems in particular, prototyping has also been criticized in terms of systems efficiency and project management and control[23,24].

Connor[18] concluded that prototyping drawbacks cannot be avoided unless some measure of structure is introduced into development. This realisation had led to hybrid approaches that mix structured techniques with prototyping[17]. Structured prototyping (SP), one such hybrid framework[25], is designed to help professionals by incorporating prototyping within structured analysis and design. The SP framwork assigns to the professional all the responsibility for analysis, design, and prototyping and assumes a traditional sharing of responsibility, whereby the user presents a 'wish list' of requests and reacts to prototype presentations by the professional, until convergence to systems specifications is reached. Such responsibility sharing, however, is in fact an opportunity loss in an environment characterized by users' sophistication. Collaboration with sophisticated users could reduce the time to produce and review professionally built prototypes.

This paper recommends a partial transfer of SP responsibilities to end-users. The extent of transfer would depend on the user's sophistication level and on the complexity of the system under development. MinUsers can sketch the initial menu interface and the input and output screens. MaxUsers might go beyond sketching the interface and participate in functional analysis, in part of the design, and in prototyping user-interface components. For noncomplex systems they might also prototype other components of the system or produce a whole UDA.

After a brief review of SP in the next section, the authors proceed to map their approach to structured methodological support for sophisticated end-users. Then some experiences with the approach are reviewed and implications for training, career planning, and further research are concluded.

STRUCTURED PROTOTYPING FOR PROFESSIONALS

In SP[25], structured techniques are integrated with prototyping; the final prototype of the system emerges from a converging series of prototypes built during analysis and design. The underlying structured development methodology is ADISSA[26], a methodology that provides a smooth transition from structured analysis[27,28] to architectural design. ADISSA covers the analysis and design stages in a unique way, fulfilling established software engineering principles, such as top-down approach, stepwise refinement, and modular design.

ADISSA includes the top seven steps shown in Figure 1 (and drawn in two columns for the purpose of later discussions):

- (1) functional analysis
- (2) menu-tree design
- (3) input/output schema design
- (4) data dictionary
- (5) transaction design
- (6) database schema design
- (7) transaction description

Next, the ADISSA steps are reviewed to set the background for presenting the prototyping steps at the bottom of Figure 1.

- (1) Functional analysis. Functional analysis is based on hierarchical modified dataflow diagrams (DFDs), enhanced with various entities (user, time, and realtime entities). Such DFDs facilitate the design steps to follow.
- (2) Menu-tree design. A menu-tree interface is derived from the hierarchy of DFDs in two main stages. In

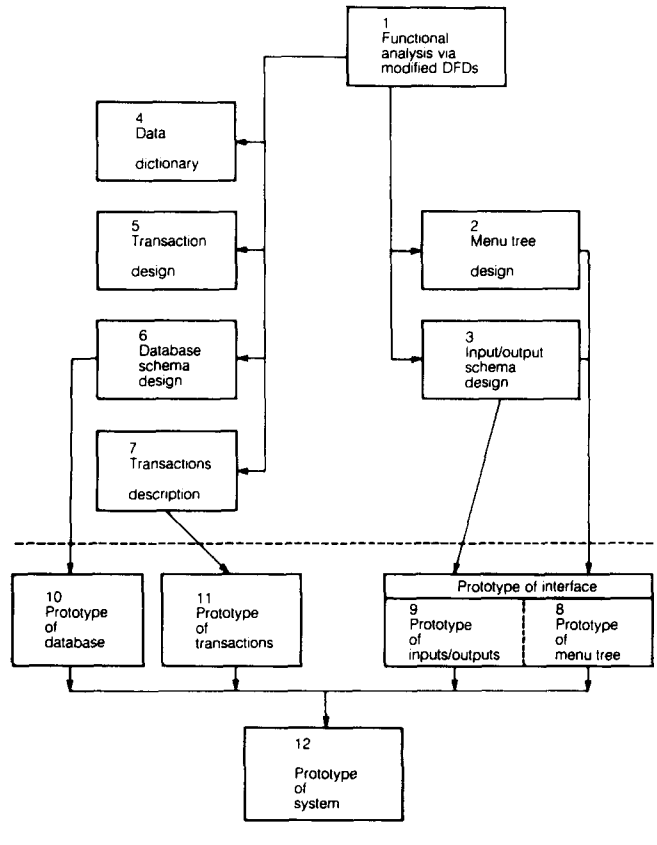

Figure 1. Structured prototyping

the first, an initial menu tree is produced automatically (i.e., algorithmically from DFDs); in the second, it is improved and refined, taking into consideration the transactions of the system (see step 5, below), until the desired menu screens are obtained.

- (3) Input/output schema design. All inputs and outputs (forms, screens, and reports) are derived from the DFDs, as based on data flows between user entities and system functions.
- (4) Data dictionary. The data dictionary is extended with respect to the menu-tree components and the transactions components (step 5).
- (5) Transactions design. The processes of the system are described as transactions, which are also derived from DFDs. Each transaction consists of one or more chained elementary functions and other DFD elements connected to them, and it has a 'triggering' entity, so that it can be triggered by appropriate selections in the menu tree.
- (6) Database schema design. Database schema design is done after transaction design. The result of this step is a relational database schema, consisting of normalized record types. Traditionally, the schema can be derived by conducting a normalization process on the DFD data stores, as in Gane and Sarson[28]. An alternative, more modern way to derive the database schema is by first creating a conceptual schema, using a semantic data model, and then derive from it normalized relations as in Shoval[29].
- (7) Transaction description. Structured descriptions of the transactions are pursued on two levels of detail. First, a top-level (general) description of the main elements of a transaction and its logic is provided. This is actually a transition from DFD bubbles to program modules. Then a detailed structured description of the transactions is provided in terms of conventional programming techniques (e.g., pseudocode), including details of functions, input/output activities (based on the designed input/output schema), and database reads/writes (based on the designed database schema). Each description facilitates the smooth transition to prototyping the transaction.

The horizontal line in Figure 1 separates ADISSA analysis and design activities (at the top) from prototyping activities (at the bottom). Though not explicitly indicated in Figure 1, there is a feedback loop between the top and bottom activities.

SP breaks prototyping into four parts (8 to 11), which are based on the earlier products of the design, as follows:

- (8) a prototype of the menu tree
- (9) a prototype of the inputs and outputs (with (8) this composes the prototype of the interface)
- (10) a prototype of the database
- (11) a prototype of the transactions

Combined, these four parts comprise part (12) a prototype of the system. Briefly, the prototypes are:

- (8) Prototype of the menu tree. The menu-tree prototype is based on the designed menu tree (2). The screen painter/editor module of an application generator is used to create the hierarchy of menu screens and adjust it to specific user demands.
- (9) Prototype of the inputs/outputs. The input/output prototype is based on the input/output schema (3). In prototyping the inputs and outputs, the builder can apply the application generator (e.g., the screen management module for screens) to facilitate creation of those inputs/outputs. At this stage, it is possible to add error checking for various input fields, help functions, etc. Of all the elements of SP, prototypes (8) and (9), comprising the interface prototype, are the most likely to be used in the production system, even if the prototype ends up as the 'throw-away' type.
- (10) Prototype of the database. The database prototype is a live and working database schema, to be created using the database management module of an application generator and its data definition language. It includes the definition of record types and their fields, key fields, field domains or types, etc. For those fields that are used as access paths, the database management environment enables a straightforward definition of secondary keys or indexes. To maintain reality for the data prototype, real data is introduced for at least some of the record types.
- (11) Prototype of the transactions. This prototype is a collection of program modules, each of which carries out a transaction. The structured descriptions of transactions (step 7) and a fourth-generation language make the programming and testing of program modules quick and manageable. Beyond the process logic, testing covers the 'reads' (retrievals) and 'writes' (updates), with respect to the database prototype, and the inputs/outputs, with respect to the input/output prototype. In a large and complex system it is not mandatory to program all transactions; rather, the prototype builder may concentrate on the most important transactions, which are associated with heavy user interaction.
- (12) Prototype of the system. The system prototype integrates all prototyped transactions into one system and also includes a menu tree that facilitates activation of the transactions. This is the phase whereby simulations with test data are studied.

Note that completion of all SP phases is not a certainty. It is a well known fact that realistic constraints may force a compromise between practice and theory. Even professionals, who preach and practice structured analysis and design, sometimes bypass structured phases for simple systems.

STRUCTURED METHODOLOGICAL SUPPORT FOR SOPHISTICATED USERS

SP integrates prototyping with the ADISSA methodology. It assumes traditional responsibility sharing between professionals and users, whereby users initially present their 'wish list' and then interactively react to the various prototypes until a consensus about system requirements is reached. The authors argue that it makes more sense to solicit active contributions from sophisticated users[1].

The approach presented in this section prescribes that sophisticated users may assume responsibilities for some or all of the tasks on the right-hand side, as well as below the horizontal line of Figure 1. Though the sharing of all responsibilities is outlined below, it is acknowledged that if both parties adopt a realistic and practical attitude, some assigned tasks could be skipped or performed superficially. Especially relevant to such relaxation are both the complexity of the sought system and the policy of DP management. While the latter is relevant mainly to professionals (and thus not elaborated on in this paper), the former pertains also to responsibilities of MaxUsers and is therefore discussed in this section.

The extent to which sophisticated users take over activities originally intended for professionals is a function of the users' sophistication: MinUsers can design the interface, i.e., the menu tree and the input/output schema (steps 2 and 3, respectively, in Figure 1), while MaxUsers can design and prototype the interface (steps 2, 3, 8, and 9). Whether MaxUsers could also analyse the system (step 1) and prototype the whole system (steps 10, 11, and 12) would depend mainly on the complexity of the system. Users of intermediate sophistication would assume more analysis, design, and prototyping responsibilities than MinUsers and less than MaxUsers. Responsibility for the remaining tasks is left in the hands of professionals. They would also guide and supervise the activities of sophisticated users, especially for complex systems.

Structured support for MinUsers

When facing MinUsers, responsibility for systems analysis would remain with professionals. In addition to the traditional dialogue between users and professionals, which is the basis for functional analysis, and in addition to user approval of the DFDs, however, MinUsers would also provide preliminary sketches of the menu tree and inputs/outputs. This would be followed by an iterative interaction between the two parties until agreement is reached: MinUsers would redesign menus, inputs, and outputs, and the professionals would refine the DFDs. The professional development team would be responsible for all other steps (4-7) on the left-hand side of Figure 1.

Once analysis and design have been completed, professionals would continue with prototyping all components of the system (Figure 1, steps 8–12). The MinUser would be allowed to navigate through the menus and comment on the functionality of the prototypes. At that time, the MinUser might also express wishes with respect to the incorporation of help screens or the ease of movement between screens. The top two lines of Table 1 reflect responsibility sharing between MinUsers and professionals.

Table 1. Structured prototyping (SP) responsibility sharing between users and information-systems (IS) professionals

Responsibility of:	SP Steps 1	2	3	4	5	6	7	8	9	10	11	12
MinUser		x	x									
IS professional	x			x	x	x	x	x	x	x	x	x
Simple systems:												
MaxUser	x	x	x					x	x	v	v	v
IS professional				x	x	x	x			v	v	v
Complex systems:												
MaxUser	v	x	x					v	v			
IS professional	v			x	x	x	x	v	v	x	x	x

x = Responsibility certain.
v = Responsibility determined according to circumstances.

MinUsers interactively apply software packages to support their professional missions (e.g., accountants use spreadsheets, librarians use database management systems). This background would not suffice for active participation in SP. MinUsers should be formally exposed to the SP process, with emphasis on tasks intended for them. In addition, they ought to be acquainted with design principles for menus, inputs, and outputs[30]. If actual interface sketching exercises for MinUsers are based on familiar software, their informal user-interface experience would facilitate fast learning (in two or three days).

Structured support for MaxUsers

The extent to which MaxUsers would assume further responsibilities, in addition to sketching menus, inputs, and outputs, is not certain and would depend on the sought system, whether simple or complex.

For simple systems, it is not clear whether the professional would be involved at all. For example, steps 4–7 could be bypassed altogether. The sought system might be so simple that neither intervention of professionals nor completion of all SP phases would be needed. Under a possible scenario for small systems, MaxUsers would be responsible for the five steps on the right-hand side of Figure 1, i.e., they would functionally analyse via DFDs, design the interface, and create the menu as well as input/output screens until those interface products are satisfactory both visually and functionally. Afterwards, they would make a presentation to professionals and a joint decision would be made about whether the system was so simple as to allow immediate prototyping of the whole system. Alternatively, professionals would take over, develop the data dictionary,

design the transactions and the database, and complete prototyping. The middle two lines of Table 1 reflect this uncertainty about responsibility sharing. (While a column marked with a 'x' indicates an unequivocal responsibility assignment, a 'v' indicates either uncertainty about the identity of the responsible party or joint responsibility.)

For complex systems, it is conceivable that MaxUsers would have difficulty in performing functional analysis and interface prototyping on their own. It is recommended therefore, that they should decide, jointly with professionals, if and to what extent they would be able to contribute to functional analysis. MaxUsers might, for example, be responsible for DFDs of a subsystem or perform functional analysis together with professionals. Once the DFDs of the system have been jointly created (and naturally agreed on), another joint decision would be if, and to what extent, MaxUsers could engage in prototyping the interface. The initial menu tree and the input/output schema might be designed by MaxUsers or be a joint venture or the two parties could also agree on only partial prototyping. The emphasis should be placed on quality improvement of these interface products, both visually and functionally. Professionals would then assume responsibility for the remaining design and prototyping steps. Responsibility sharing between MaxUsers and professionals for complex systems is reflected in the bottom two lines of Table 1.

Training that would enable MaxUsers to be involved in the above-described role should be more extensive than MinUser training. Their training ought to go beyond SP review and interface design and include some theory and experience in systems analysis and SP methodology. This kind of training will not turn MaxUsers into professionals, and for best performance they would need professional guidance and supervision. MaxUsers' experience in spreadsheet and database application development guarantees that training would not last more than one or two weeks.

CASE STUDY OF RESPONSIBILITY SHARING IN SP

This section describes one of the many possible scenarios of the SP process involving sophisticated users. The case study deals with a marketing firm, Medical Supplies Corporation (MSC), an importer and distributor of medical equipment. The strategic emphasis at MSC is on marketing, sales, and service. Early in 1988, among other revolutionary means considered for better realisation of the marketing potential of MSC, top management opted to seek actively the computerization of the marketing function at MSC to replace the old manual sales processing.

An IBM System 36 and software was acquired in 1987. By April 1988, the accounting and inventory modules became fully operational. No progress was made, however, toward full activation of the modules that deal with sales, purchasing, and customer service. The software did not fit MSC's requirements. Furthermore, some of the employees found the software unfriendly and the interface tedious. They also complained that it entailed a cumbersome dependence on manual inventory and customer books, thus diminishing the advantages of computerization.

Top management was deeply concerned with the partial implementation of the software for the IBM System 36, because the unimplemented modules were perceived to be critical to the strategic advantage of MSC. A consultant, who was hired to do a feasibility study, interviewed MSC employees, reviewed existing forms and reports, and analysed the MSC marketing infrastructure *vis-à-vis* software offered by vendors. Based on the study's findings, top management concluded that no software package on the market could meet the firm's strategic goals and was in favour of tailor-made development.

Top management sought to prototype partially the marketing system before distribution of a prototype among vendors as an integral part of the request for proposal. The prototyping assignment was allocated to a part-time sales person (at the time also a senior student at an Industrial Engineering and Management program). That sales person recruited a fellow student to join him on the project as a summer job. The two functioned as a team and documented their prototyping activities for a final graduation project at university.

The course of events provided an opportunity for realistic testing of collaborative responsibility sharing between a sophisticated user and a professional in structured support methodology. The MSC sales person (hereafter referred to as the sophisticated user) certainly met the specifications of a user: he knew the marketing function at MSC inside out and understood very well MSC operations and procedures. In terms of DP sophistication, he exceeded the authors' description of a MinUser but was not yet a MaxUser: he had extensive training as an undergraduate student in dBase and Lotus and had ample opportunity for actual application development at work. He also happened to acquire proper SP training through auditing lectures in a systems analysis and design course and reading publications on ADISSA and SP. The other team member (hereafter referred to as the professional) came close to meeting the specifications of a professional: he gained (limited) practical DP experience on previous summer jobs, majored in information systems, and was knowledgeable in ADISSA, SP, programming, and database management. Furthermore, he was objective as he had had no previous dealings with MSC in general and marketing in particular. The following paragraphs describe the collaboration between the students.

(1) Functional analysis

Initially, the sophisticated user analysed the requirements and created a hierarchical set of DFDs, and the professional was available for consultation, e.g., to identify logical inconsistencies. Later on, however, when DFD updates were required, they preferred team work.

Three of the DFDs, representing a small portion of the

system and slightly modified for this paper (e.g., for the sake of simplicity, only Figure 4 is labelled with data flows), are presented in Figures 2–4. Figure 2, DFD-0, contains the six general functions of the system (denoted by concentric circles), the main external entities—user entities (denoted by rectangles) and time entities (denoted by triangles), the data stores, and the data flows among them. Figure 3, DFD-2, details the general function 'Orders and supplies' and includes two general functions and four elementary functions. Figure 4, DFD-2.1, details 'Order entry' and includes nine elementary functions.

(2) Menu-tree design

The initial menus were derived from the DFDs by the sophisticated user. Modifications and improvements due to the transaction design, however, required the professional to be involved. This will be discussed below in step 5. A close working arrangement with the professional helped the sophisticated user to cope with the medium complexity of the system.

The sophisticated user reviewed the DFD hierarchy to find DFDs with functions that are connected to external

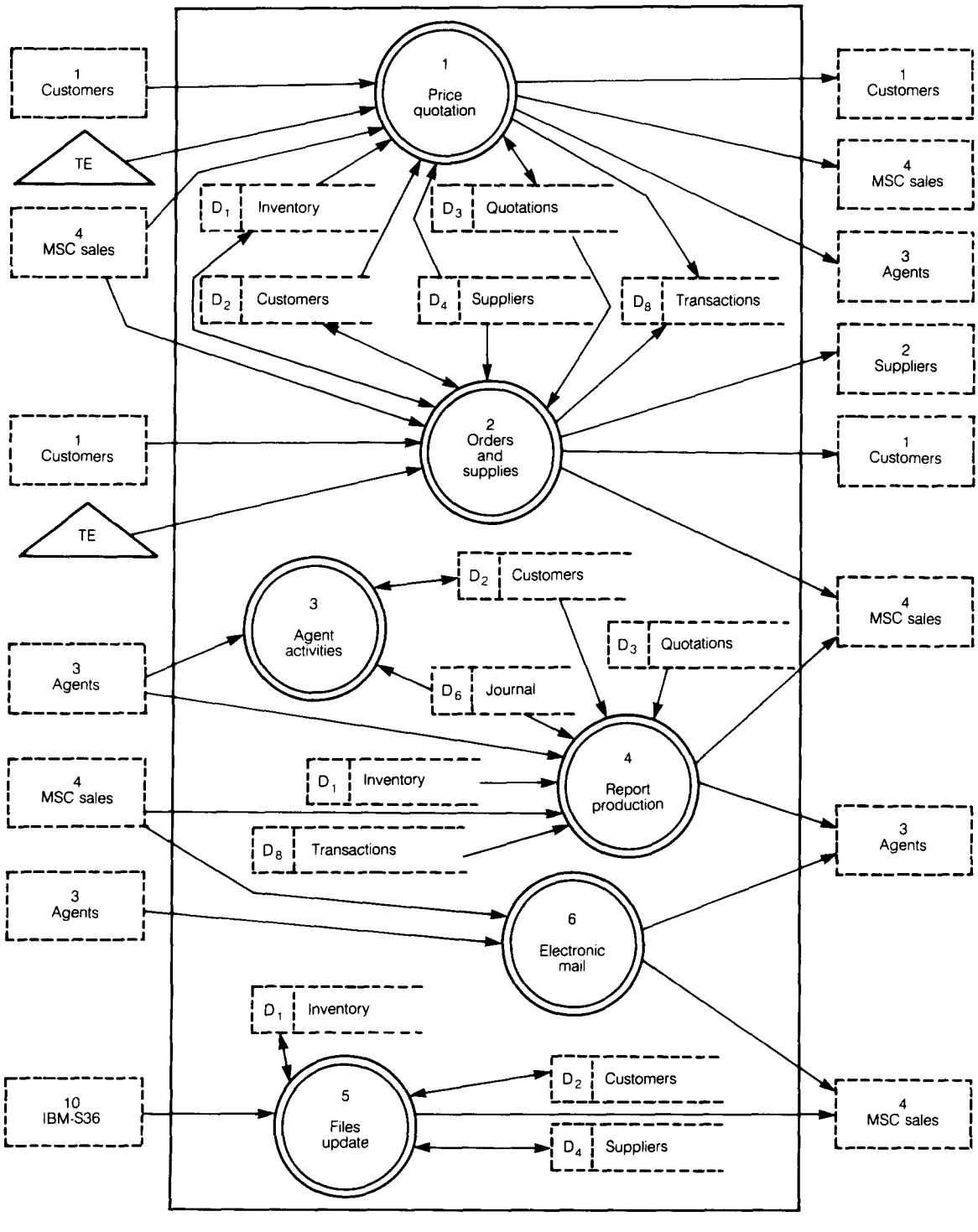

Figure 2. MSC case study - DFD-0

entities. For each such DFD, a menu screen was defined to include a line for every function linked to an entity. Data flows between entities and general functions were translated into selection lines, whereas data flows between entities and elementary functions were translated into terminal lines. The set of derived menus yielded an initial menu tree that matched the DFD hierarchy. Figure 5 presents part of the initial menu tree for this portion of the system. In the root DFD (DFD-0 of Figure 2) all six general functions are connected to external entities. Correspondingly, the root menu screen in Figure 5 includes six selection lines. In DFD-2 there are two general functions and four elementary functions connected to external entities; therefore, the corresponding menu in Figure 5 contains six lines, four of which are terminal (hence the 'T' mark). The next menu level of Figure 5 contains four terminal lines that reflect the four elementary functions of DFD-2.1 connected to external entities. At the menu-design stage the selection and terminal lines of the menu screens are labelled in a manner consistent with the labelling of the corresponding functions, but these may change later on.

(3) Input/output schema design

From this point on, the team focused on three functions perceived by management to be worthy of prototyping, including 'Orders and supplies'. For these, the sophisticated user identified data flows that connect external entities and elementary functions in the DFDs. A data flow from an entity to an elementary function reflects data input (e.g., an input screen or a form). Similarly, a data flow

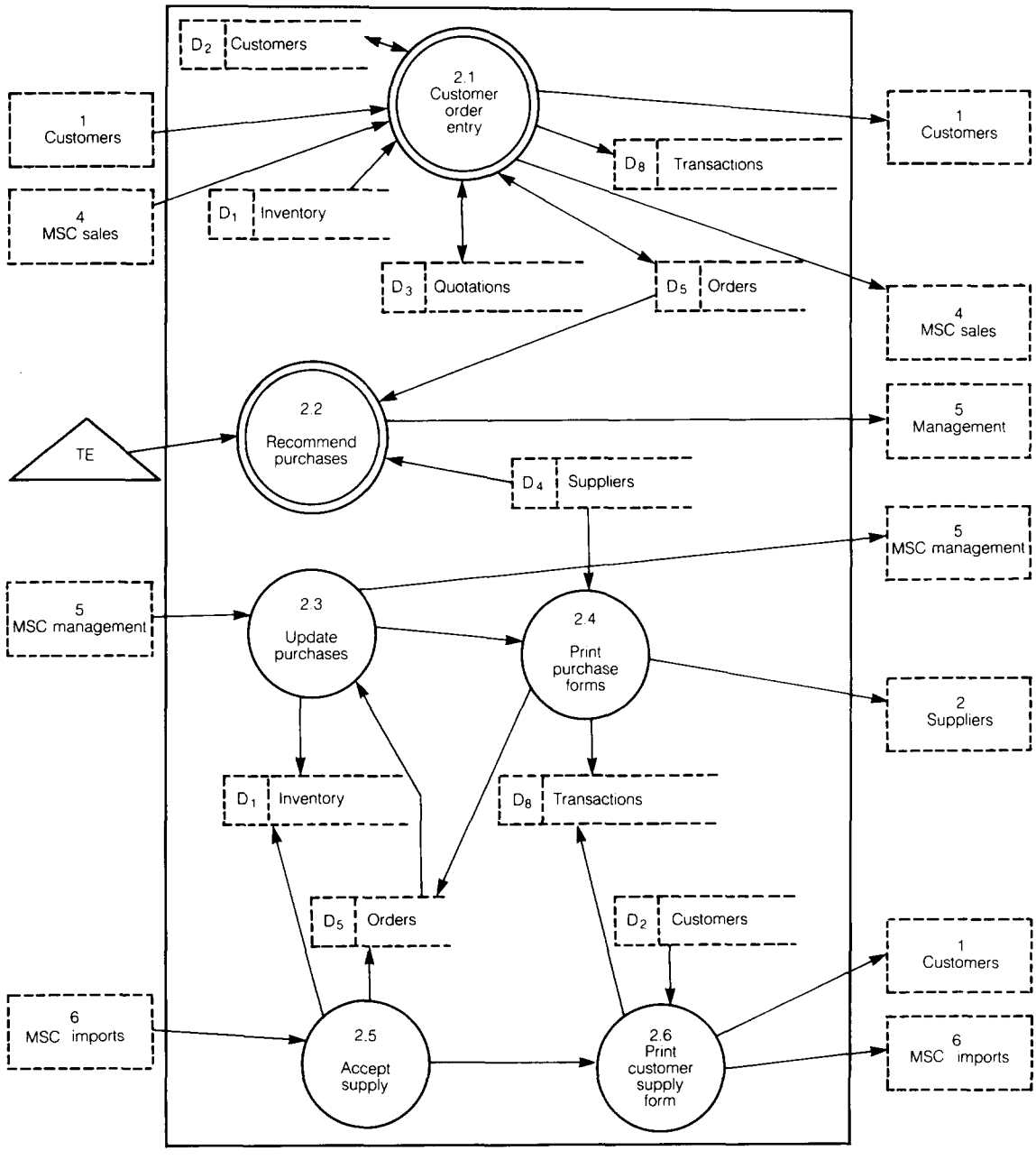

Figure 3. MSC case study - DFD-2

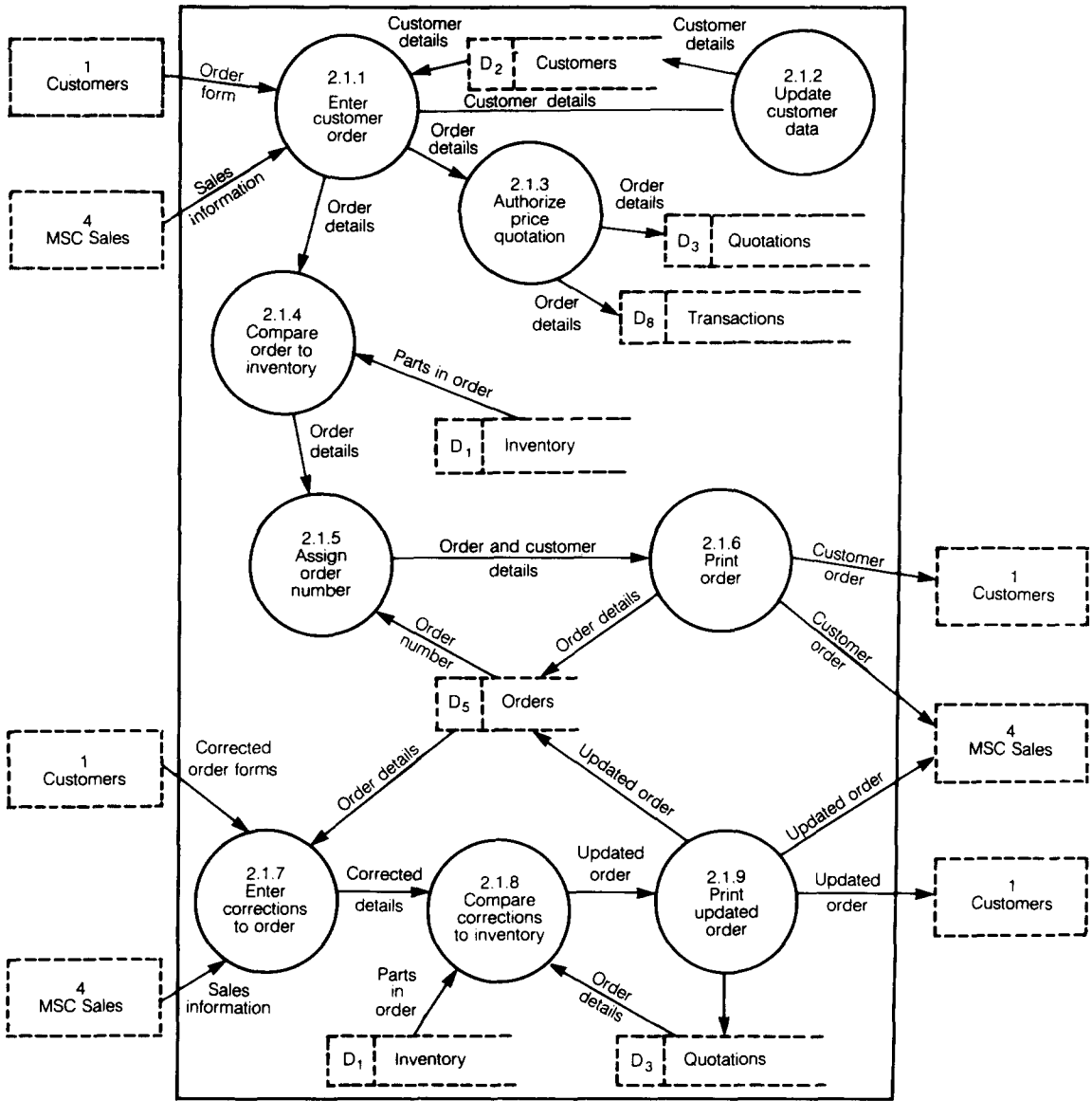

Figure 4. MSC case study - DFD-2.1

from an elementary function to an entity is associated with an output screen or a printed report. All inputs and outputs comprise the input/output schema of the system. Its design by the sophisticated user included a decision about the media and a sketch of the data elements to be included.

DFD-2.1 (see Figure 4) contains a data flow from a customer to function 2.1.1. The sophisticated user assigned an input screen that would contain various data elements that comprise the customer order. This screen would be presented to the system operator on selection of line 2.1.1 in the bottom menu (see Figure 5). The operator would be required to fill the screen and possibly to express a desire for changes. The data elements considered at this point were selected by the sophisticated user. Later, the professional created the data dictionary based on these elements.

Theoretically, this stage involved only sketching of the data elements to be entered, leaving actual screen layout and format to step 8. In practice, however, transition to step 8 was immediate because the professional had made progress on the next three steps and encouraged the sophisticated user to prototype the interface soon after initial sketching was completed.

(4) Data dictionary

The data dictionary was compiled by the professional in cooperation with the sophisticated user. Building of the data dictionary was facilitated by ADD (ADISSA Data Dictionary), one of the software tools that support the ADISSA methodology[31]. Without going into detail, it is noted that definitions of some of the data elements were determined as based on the input/output screens designed earlier by the sophisticated user.

(5) Transaction design

Transactions were designed by the professional according to the ADISSA methodology. Four of these transactions

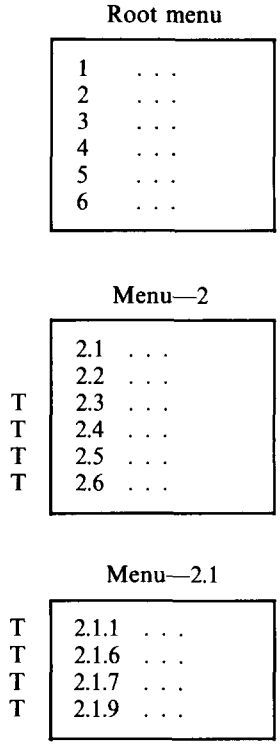

Figure 5. Initial (partial) menu tree

are now mentioned. In DFD-2 (see Figure 3) there is a transaction that consists of functions 2.3 and 2.4 (and other DFD elements connected to them) and another that consists of functions 2.5 and 2.6. In DFD-2.1 (see Figure 4) one transaction consists of six functions (2.1.1 to 2.1.6) and the other of three functions (2.1.7 to 2.1.9).

While the professional designed the transactions, he improved the menu tree together with the sophisticated user, so that it would correspond to the designed transactions. Generally, in ADISSA, every user transaction should be activated by a single triggering line in the menu tree, but sometimes the initial menu tree contains multiple triggering lines for the same transaction. This was the case, for example, in the transaction that consists of functions 2.3 and 2.4, which had two triggering lines in menu screen 2 (see Figure 5) because both functions 2.3 and 2.4 are connected to external entities; the two lines were united. Similarly, the line pairs 2.5/2.6, 2.1.1/2.1.6, and 2.1.7/2.1.9 were united as well.

(6) Database schema design

The professional was responsible for the database schema design: every data store in the DFDs was normalized, resulting in a set of normal-form relations. All relations that result from the data stores comprise the database schema. The professional also defined access paths by reviewing the 'reads' and 'writes' in the transactions. The sophisticated user was hardly involved in this stage.

(7) Description of transactions

This stage, which enables transition from the transaction diagrams to descriptions of their underlying logic process, was carried out by the professional in two parts. First, he created a top-level (general) description of the transactions, and then he detailed these descriptions. The top-level descriptions included activities on the main elements of a transaction, such as: 'Execute', 'Input-from entity', 'Output-to entity', 'Read-from data store', and 'Write-to data store'. For example:

Begin Transaction 2.1.7-2.1.9
Input from 1 (Customers): corrected order form
Read from D5 (Orders): order details
Input from 4 (MSC-sales): order comments
Execute F2.1.7 (Enter corrections to order)
Move to 2.1.8
Read from D1 (Inventory): parts in order details
Read from D3 (Quotations): order details
Execute F2.1.8 (Compare corrections to inventory)
Move to 2.1.9
Execute F2.1.9: (Print updated order)
Write to D5 (orders): updated order
Write to D3 (Quotations): updated order details
Output to 1(Customers): updated order
Output to 4 (MSC-sales): updated order
End

Each top-level description was then detailed using one of the following structured programming techniques: structured English, structured flowcharts, decision tables, and decision trees. These structured descriptions were instrumental during the writing of code in step 11 below.

(8) Prototype of menu tree

Prototyping the interface by the sophisticated user, at the bottom right of Figure 1, was a direct continuum of the menu-tree design and the input/output schema design. The sophisticated user, using dBase, not only prototyped the menu tree but also introduced help functions in this stage.

(9) Prototype of inputs/outputs

In prototypng the interface, the sophisticated user applied the dBase form and report creation facilities to generate the inputs and outputs. For example, from the sketch of the price quotation input screen of the input/output schema, he proceeded to format the screen.

(10) Prototype of database

The professional created dBase files for the relations designed earlier. Together with the sophisticated user he entered a few actual data records into the various relations and tried out some common database activities (e.g., append, edit, replace, and list).

(11) Prototype of transactions

The professional used the dBase command language to write programs for each of the transactions described

previously (in step 7). In each program he related the various 'read's and 'write's to specific relations in the database (step 10) and also included the relevant input and output screens (step 9). These programs were individually run by the sophisticated user, who requested minor modifications.

(12) Prototype of system

This stage integrates the individual program modules developed for each transaction into one system, using the menu-tree prototype. This integration is done to ensure that selection of a triggering line in the menus activates the corresponding transaction program. Following the integration, the sophisticated user collaborated with the professional to add error messages, to modify inputs and outputs, etc. When both team members were satisfied with the prototype they presented it to other potential users, received feedback, and performed additional modifications. Modifications were not limited to the prototype elements of SP but were also carried out to the analysis and design products, thus keeping documentation up-to-date at all times. The prototyping team then prepared the documents necessary for approaching potential software houses that would be interested in bidding for the project.

Table 2 summarizes the sharing of responsibilities in the MSC case study. Collaboration between the sophisticated user and the professional throughout the SP steps was intense and, unlike Table 1, which assigns to one party total responsibility for each step, in some SP steps one team member was dominant and the other assumed a consulting role (as indicated in Table 2 with a 'd' and a 'c', respectively).

CONCLUSIONS

The MSC case constituted an intermediate scenario in terms of both system complexity and user sophistication and demonstrated the iterative nature of the approach and the underlined feedback loop. The collaboration reviewed can be presented in terms of two nested loops: a loop of prototyping activities nested within a loop of analysis and design. In each loop, intensive collaboration can be expected, whereby division of responsibilities is consistent with the separation between the external and internal architectures. In the analysis and design loop, the sophisticated user would be active on DFDs, menu tree, inputs, and outputs (steps 1, 2, and 3), while the professional would design the transactions and the database, create a data dictionary, and describe the transactions (steps 4, 5, 6, and 7). A similar division of roles would be in the prototyping loop; the menu screens, inputs, and outputs (steps 8 and 9) would be prototyped by the user, and the rest (steps 10, 11, and 12) by the professional.

Table 2. Structured prototyping (SP) responsibility sharing in MSC case study

	SP Steps											
Responsibility of:	1	2	3	4	5	6	7	8	9	10	11	12
Sophisticated user	d	d	x		c			d	x			c
IS professional	c	c		x	d	x	x	c		x	x	d

x = Responsibility certain.
c = Consulting role under shared responsibility.
d = Dominant role under shared responsibility.

Beyond the qualitative description of the case study, quantitative data collected provide an additional insight to the proposed approach. Altogether, 90 alterations of the DFDs were made, and one DFD was added. These, as well as the addition of seven transactions and the many more modifications to other transactions, resulted from hand-on simulations by potential users. The total number of data stores remained constant, but the number of data files went down from 14 to 12. About 25 changes to database fields were made, resulting from adding fields and values not identified before prototyping. Finally, many 'cosmetic' alterations took place, including the addition of help, error, and system messages and altering the format of menus, inputs, and outputs.

The sophisticated user and the professional invested about 180 hours in the project. In addition, other users (e.g., sales personnel) invested about 20 hours in seven meetings. Although it was clear that throughout the information-systems history at MSC never were so many hours invested before software acquisition, there was agreement among all involved that the 200 man-hours invested in a 'throw-away' prototype was a small price to pay for the information gained and for the reduction in uncertainty and risk. All involved seemed hopeful that on development of the production system the frustrations and implementation failures MSC had previously experienced would be avoided.

The authors' observations on structured methodological support for sophisticated users, though preliminary, point to the following propositions:

- Sophisticated users are resourceful in applying fourth-generation languages and after presentation of the methodology need little help from professionals. The inherent structure helps them to improve UDAs and they appreciate the capacity of structured tools to put development within a manageable framework. They help relay system functionality details to other users well in advance. The better communication between the users and the professionals, the collaboration, and the cooperation promote user satisfaction.
- Professionals appreciate the fast convergence to requirements, the quality of systems, and the increased contribution of users.
- The structured framework entails extra time investment by sophisticated users. Small UDAs might take longer to develop, and functional analysis of the system and design of the user interface for large systems might demand more time from sophisticated users than would

be the case for conventional development. Training them to apply the methodology is easy, fast, and inexpensive. They can be taught and practised either in an academic course or in a business setting, within a workshop of a week or two.

Further research and real-world evidence on small and large systems is needed to verify the benefits and assess the limitations of the methodology. Are conflicts between users and professionals eliminated, or are they replaced by new conflicts? Are there development situations whereby the benefit-cost ratio is more favourable than in other situations? Furthermore, an investigation of the managerial implications of the approach is needed. For example, how can mixed teams of professionals and sophisticated users be promoted and encouraged? Such future research would have to be interdisciplinary.

REFERENCES

1 **Pliskin, N and Shoval P** 'End user prototyping: sophisticated users supporting system development' *Database* Vol 18 No 4 (1987) pp 7–17

2 **Huff, S L, Munro, M C and Martin, B H** 'Growth stages of end user computing' *Commun. ACM* Vol 31 No 5 (1988) pp 542–551

3 **Rivard, S and Huff, S L** 'Factors of success for end user computing' *Commun. ACM* Vol 31 No 5 (1988) pp 552–561

4 **Ahituv, N, Baroudi, J J and Ronen, B** 'The characteristics of microcomputer support personnel: a normative perspective' *Human Syst. Manage.* Vol 7 (1988) pp 321–331

5 **Benson, D H** 'A field-study of end user computing: findings and issues' *MIS Q.* Vol 7 No 4 (1983) pp 35–45

6 **Lee, D M S** 'Usage patterns and sources of assistance for personal computer users' *MIS Q.* Vol 10 No 4 (1986) pp 313–326

7 **Martin, J** *Application development without programmers* Prentice Hall, Englewood Cliffs, NJ, USA (1982)

8 **Pliskin, N** 'People resource management: guidelines for effective introduction of office technology' *J. Inf. Syst. Manage.* Vol 6 No 2 (1989) pp 51–58

9 **Rockart, J F and Flannery, L S** 'The management of end user computing' *Commun. ACM* Vol 26 No 10 (1983) pp 776–784

10 **Sumner, M R** 'How should applications be developed? An analysis of traditional, user, and microcomputer development approaches' *Database* Vol 17 No 1 (1985) pp 25–34

11 **Michelsen, K** 'Micro application development' *Datamation* Vol 32 No 8 (1986) pp 96–98

12 **Rivard, S and Huff, S L** 'User developed applications: evaluation of success from the DP department perspective' *MIS Q.* Vol 8 No 1 (1984) pp 39–49

13 **McFarlan, F W** 'Portfolio approach to information systems' *Harvard Bus. Rev.* Vol 59 No 5 (1981) pp 142–150

14 **Alavi, M** 'An assessment of the prototyping approach to information systems development' *Commun. ACM* Vol 27 No 6 (1985) pp 556–563

15 **Berrisford, T R and Wetherbe, J C** 'Heuristic development: a redesign of systems design' *MIS Q.* Vol 3 No 1 (1979) pp 11–19

16 **Bohr, B H** *Application prototyping: a requirements definition strategy for the 80s* John Wiley, New York, NY, USA (1984)

17 **Burns, R N and Dennis, A R** 'Selecting the appropriate application development methodology' *Database* Vol 17 No 1 (1986) pp 19–23

18 **Connor, D** *Information system specification and design road map* Prentice-Hall, Englewood Cliffs, NJ, USA (1985)

19 **Earl, M J** 'Prototype systems for accounting, information, and control' *Database* Vol 13 No 2–3 (1982) pp 39–46

20 **Kraushaar, I M and Shirland, L E** 'A prototyping method for applications development by end users and information systems specialists' *MIS Q.* Vol 9 No 3 (1985) pp 189–196

21 **Mason, R E A and Carey, T T** 'Prototyping interactive information systems' *Commun. ACM* Vol 26 No 5 (1983) pp 347–354

22 **Naumann, J D and Jenkins, A M** 'Prototyping: the new paradigm for systems development' *MIS Q.* Vol 6 No 3 (1982) pp 29–44

23 **Janson, M A and Smith, L D** 'Prototyping for systems development: a critical approach' *MIS Q.* Vol 9 No 4 (1985) pp 305–316

24 **Jenkins, A M** 'Prototyping: a methodology for the design and development of applications systems' *Spectrum* Vol 2 No 2 (1985)

25 **Shoval, P and Pliskin, N** 'Structured prototyping: integrating prototyping into structured system development' *Inf. Manage.* Vol 14 (1988) pp 19–30

26 **Shoval, P** 'ADISSA: architectural design of information systems based on structured analysis' *Inf. Syst.* Vol 13 No 2 (1988) pp 193–210

27 **DeMarco, T** *Structured analysis and system specification* Prentice-Hall, Englewood Cliffs, NJ, USA (1982)

28 **Gane, C P and Sarson, T** *Structured systems analysis: tools and techniques* Prentice-Hall, Englewood Cliffs, NJ, USA (1979)

29 **Shoval, P** 'Essential information structure diagrams and database schema design' *Inf. Syst.* Vol 10 No 4 (1985) pp 417–423

30 **Shneiderman, B** *Designing the user interface: strategies for effective human-computer interaction* Addison-Wesley, Reading, MA, USA (1987)

31 **Shoval, P** 'ADISSA tools' in *Proc. Fourth Israel Conf. Computer Systems and Software Engineering* (1989) pp 153–157