Developments in Business Simulation & Experiential Exercises, Volume 13, 1986

THE DESIGN OF SIMULATION MODELS USING THOUGHT ORGANIZERS

F. Paul Fuhs, Virginia Commonwealth University

ABSTRACT

A new class of software has arisen. It has its evolutionary roots in word processors, but far surpasses word processing functional capabilities to manipulate the structure of ideas during the development process of written text. This class of software is called Thought Organizers. In this paper we extend the use of thought organizers to the development of simulation models. We demonstrate why and how these packages are appropriate for the design and implementation of simulation models.

INTRODUCTION

The development of computerized software for users can be divided into two classes; application systems and simulation models. The development of most application systems is a team effort, involving the management and coordination of a large number of analysts and programmers. Simulation models, on the other hand, tend to be developed either by professional software houses, again with the use of large teams of people, and with hefty research and development budgets or by individual designers whose models are smaller in scope and whose model use is mostly for private consumption. This paper addresses this latter population of simulation model developers.

The development process of both application systems and simulation models includes design stages and implementation stages, beginning with specification of model objectives and ending with computer code expressed in some high level language. The design stages for a simulation model include the determination of:

1. The overall objective of the model.
2. The sub-objectives of the model.
3. A realistic setting for the model.
4. The general scope of the model within that set ting.
5. The constraints on model development time.
6. The constraints on computer resources like CPU time and equipment. [1]
7. The constraints on participants' time to use the model.
8. The entities, attributes and relationships within the model.
9. The role of the user in relation to the model (passive observer, active participator).
10. Which entities, attributes, and relationships should be transparent to the user and which should be explicitly described either before the model is run, during its execution, or after execution.
11. User interface points within the model.
12. User input content and format.
13. Model output content and format.

The implementation stages include:

1. The selection of an appropriate simulation language to express the model.
2. The translation of the conceptual design of the model into functional specifications.
3. The decomposition of the functional specifications into computer program modules.
4. The coding of each module into the simulation language.

There are a number of tools available for general software development. The National Bureau of Standards publication on Software Development Tools divides software tools into the following categories:

1. Software Management, Control and Maintenance Tools
2. Software Modeling and Simulation Tools
3. Requirements/Design Specification and Analysis Tools
4. Program Construction and Generation Tools
5. Source Program Analysis and Testing Tools
6. Software Support Systems/ Programming Environment Tools

The first category is primarily for Data Processing Departments confronting the problem of controlling hundreds of computer programs. The fourth and fifth categories are primarily devoted to the professional programmer and these tools are not geared to the expression of simulation models that are event driven. Of possible interest to the development of simulation models are categories 2, 3, and 6.

Applicability of Design Tools to Small-scale Model Development

In these three categories of software tools there are many tools for the development of entire computer systems and for large scale modelling, but there is no noteworthy single tool for small scale simulation model development. We differentiate between small- scale simulation development involving an individual designer from large scale development because large scale development involves a separate class of problems related to communication and coordination between team members. Much research has already been devoted to these type problems.

Tools now exist to handle large scale application system development. However, these tools are not appropriate for small scale simulation modeling for a number of reasons. The first reason is cost. Many of these development tools cost tens and even hundreds of thousands of dollars. This expense is usually only warranted when the systems being developed are large scale. Many of these tools are aimed heavily at project management, which is inappropriate for small- scale development. Also, many are complicated to learn. Tools like SCERT which model large computer systems can take months to learn and may require specialized training under a vendor. Many such tools as PSL/PSA have their own built-in computer languages that must be learned before being useful. Here the expense is not only in dollars, but time invested in the tool. An additional reason why some of these large-scale tools are inappropriate is that they themselves are simulation models that require the user to master some of the intricacies of the model and the database with which it works. Again SCERT is an example of this type of tool. Many of these large scale development aids, when examined in detail, are not generally applicable to a wide variety of systems development. In reviewing the tools under the category “Software Modeling and Simulation Tools” we find many of these tools are restricted to the modeling of entire computer systems. One example of this class of development tools is “Performance Oriented Design”, (POD) that provides designers and implementers with early warning of potential performance problems and focuses attention on critical problem areas. This does help to reduce total development time and cost, but is of no help to the small scale simulation model designer, who is not primarily interested in execution time performance.

Many of these tools are restricted in distribution. For example, “The Design Analysis System”, (DAS) is easy to learn, is a general system modelling tool with interactive graphics and automatic model generation as its key features. It even contains an automatic documentation support system. Nevertheless, it is restricted in distribution primarily to Hughes subcontractors. Another reason is lack of portability. By this we mean the tool has been developed to run on only a restricted family of computers. The tool, “Design-Aids for Real-time Systems,” (DARTS) is an example of a development tool that is both distribution restrictive and computer hardware restrictive. For example, it is for in-house use only at the Charles Stark Draper Laboratory and it is written in PL/1.

The problem before us is the fact that the development of simulation models is a time consuming process. There is a need for a single coherent computerized tool to support the entire development process of simulation models, when these models are to be developed by individual persons rather than by large groups of designers.

The Thinking Process within the Design Process

When we sit down with a blank piece of paper or a blank CRT screen before us, what is the thinking process for system (or model) design? Structured Design techniques dictate that we begin the design process on the highest level of generalization, the highest level of abstraction. From there the designer refines the specification adding lower levels of detail until the design is complete. here the developmental process is assumed to be both hierarchical and linear. It is hierarchical in that multiple levels of abstraction are recognized as the basic framework for systems design. Structured design techniques and especially top-down systems design implies a linear developmental process with little or no iteration. One marches through the design from top to bottom. After the highest levels of specification have been layed out, one applies primarily a deductive approach to the design process. Modifications are made to higher level specifications only if absolutely necessary. Iteration is judged to be aberration.

The Expression of Creativity in the Design Process

There are a number of well documented studies in both Cognitive Psychology and Managerial Behavior to support the view that designers are not ‘odd” or mentally deficient in pursuing systems development iteratively [3,41. By iteration in systems design we mean the changing of design specifications on all levels of the design, not just patching at a single level of abstraction. Iteration is for most designers the normal modus operandi of the thinking process, whether they are developing actual systems or simulation models. Our minds dynamically follow conscious and subconscious association trails, finding new relationships, developing new concepts. and continually applying the principles of induction and deduction. Through induction and abstraction we make system specification changes on the higher, more general levels of the simulation model. Through deduction and lower levels of abstraction our minds flesh out specification details. The important point is that in the design process we simultaneously flesh out the design both horizontally and vertically. A horizontal modification is a change on the same level of abstraction. A vertical modification is one that uses induction or deduction, thereby changing ideas or specification topics on different abstraction levels.

The introduction into the system life cycle within data processing over the past few years of prototyping of systems highlights the gradual acceptance of developmental iteration into systems design methodology. As Ken Orr has said, “Iterative development usually means that the user develops a prototype of the desired system until he is satisfied with it; then the prototype is tuned until it performs satisfactorily or until the user identifies additional requirements” L51. User requirements are no longer gathered at a single developmental stage within systems design, the system built and then handed over to the users, no matter how “well structured.” This is especially true of large-scale systems. The recognition of iteration is important to systems design in general and critical for simulation model design, as we will now discuss.

Differences in Designing Actual Systems and Simulation Models

There is a fundamental difference between the design of large-scale production type application systems and simulation model design. In actual (non-simulation) application systems that will be used to support business decision making, the users data requirements impose strict limitations on the creativity of the designer. These requirements pre-define the output content, the output format, the processing needed to be accomplished, and the data to be stored. In this design environment structured analysis and design techniques have a higher imperative and iterative design a lower one. High level abstracted processes like “order entry sub-system” and associated data structures are usually easier to discover in the analysis part of the design and obviously can not be allowed to fall through the cracks by being overlooked. The sign-off procedures at specific points in the design procedure by various levels of management whereby they agree to freezing the design, attest to the reluctance to include iteration in the design procedure. The costs of retrofitting the design on the general level after detailed levels have been designed excludes any emphasis on iterative design.

However, in the design of small-scale simulation models iterative design takes on more importance. The reason for this is that this type of design is more open ended as to objectives, model structure, inputs, outputs, and processing. The entities, attributes, relationships, events, and processes used in a simulation model are not as specifically pre-determined by user requirements as application programs. Simulation model designers are constrained by the reality that they model and the level of competency of the users. However, under this broad umbrella the simulation designer has much freedom of expression. Since a model, by definition, is an abstraction from reality, there is much freedom for creativity.

The Affect of this Creativity on Simulation Model Design

We do not mean to imply that structured design is "bad". We do imply that structured design alone is too restrictive. Structured design must be complemented with the realization that the human mind doesn't create models of reality in a linear fashion. The hierarchical aspect of the Structured design approach does give us a framework for developing models. But its linear aspect is too restrictive and unproductive. To follow blindly the linearity aspect puts simulation modelers in a Procrustean bed. This does not change our creative processes nor overcome our mental limitations. Therefore, what is good for the systems design goose may not be good for the simulation model gander. This difference in design methodology further restricts the use of large-scale design tools for small-scale simulation modeling.

Functional Criteria for a Design Tool

An acceptable design tool for simulation model development must be hierarchical to include the top-down approach, yet non-linear in its use. Such a tool must support at any given point of time in the development process a tentative hierarchical structure that can be modified rapidly on any abstraction level. One should be able to flesh out the details of an already given topic or idea at one minute and add higher level ideas the next minute to another part of the evolving model. Specifically, the tool should be able to record and keep organized the infusion of new entities, attributes, relationships, and processes into the creative design of a simulation model. The tool must be able to keep up with the creative flow of changes to the design as the designer is interactively creating a simulation model at a CRT. The tool must be able to do away with traditional pencil and paper. An assumption we make is that the tool will be used by a person who is comfortable with a computer and can be creative while working with it interactively.

THOUGHT ORGANIZERS

A Thought organizer "is a tool that allows users to focus their thinking on more concepts, to examine more alternatives and to create more idea relationships --a thinking aid that lets users easily shape and reshape ideas." L61. Like spreadsheets and desk organizer programs as Sidekick the birth of thought organizers arrived on micro-computers. The first such package was ThinkTank that came onto the market in 1983. it was heralded as a new class of software, one that had as its forebear word processing, but far surpassing these packages. They are called brainstorming tools, thought processors, or thought organizers. Their purpose is to help users organize their thinking and thereby be more creative in a shorter period of time. They are to ideas what spreadsheets are to numbers. The most recognized packages on the market today are MaxThink, ThinkTank, THOR, and the Idea Processor. These packages range in price from $60.00 to
$300.00.

Characteristics of Thought Organizers

The following general characteristics of Thought Organizers are listed without reference to any specific software package, but are given to show the reader some of the functional capabilities of this group of software. The lowest level of functionality is their ability to perform text editing as word processors can, since word processing is a sub-set of their capabilities. While MaxThink, for example, can do what most word processors can do and more, editing functions, however, do vary among the packages.

The word processing capabilities within the Thought Organizers include functions internal to text and external to text. External functions include getting on-line help, performing operating system commands from within the package, having file handling features like changing file names, erasing, and saving files, reading in other text files or parts of them to join with already existing work. Internal text functions include being able to skip to the beginning or end of a block of text, delete words and phrases, find words and phrases within the text, replace words and phrases of text, move text from one place to another, copy text from one place to another, insert text within the body of already existing text, add text to the end of existing text, format output and be able to view how the printed output will look before it is printed. In addition, one looks for the capability to "undo" one or more functions when applied incorrectly.

Deficiencies with Word Processors for Model Development

We have pointed out that two of the primary characteristics of simulation development are the continual maintenance of a hierarchical structure to the model and iterative design. Word processors are not well suited for hierarchical organization. Word Processors are not truly hierarchical. They can manifest a single hierarchical structure through indentation of text, but this structure does not support the dynamic changes in structure that simulation model design demands. For example, if the designer wishes to insert a topic at a specific level in a hierarchy in Word- star, after inserting the text, he is faced with two problems. All subordinate levels in the hierarchy are no longer properly indented and if he has numbered the topics or paragraphs for reference, the number sequence is now out of order. There is no automatic renumbering of ideas or topics in Wordstar. Another problem with word processors is that all of the levels are displayed on the CRT at the same time, unless one were to go to great lengths to set up subordinate levels as separate files. This does not help thought organization, since the designer finds it difficult to distinguish the forest from the trees. An additional purpose of a hierarchical structure is to allow the designer to concentrate at any one time on a few levels within a small part of the model, not to continually have to view the entire model. This lack of hierarchical structure in word processors is also a hindrance when one considers the relatively long time it takes to navigate through a large body of text. Most word processors jump through text by moving through individual lines at a time, or by paging forward or backward, or by jumping to specific markers embedded in the text. An exception to this is when one wishes to go to the beginning or the end of the text. Nevertheless, what if the simulation designer wants to immediately jump to topic 6 on level 3.7.8.7? Word Processors are of no help here. A thought processor, built to handle true hierarchical structures, can navigate to specific hierarchical levels must faster. The reason for this is that they internally maintain linked list structures to support the hierarchy of ideas or topics.

The movement of blocks of text from one place to another with word processors is clumsy and easily tends to break our train of thought. Consider the difference between Wordstar and MaxThink when both are to move a block of text located in the middle of the screen to a specific location elsewhere. In Wordstar we have to move the cursor to the beginning of the block, type two characters, a control character and the letter B, then move the cursor to the end of the block, again typing two characters, a control character and the letter K. Then we have to move the cursor down the text to the location we wish to move it, and finally we type in two characters to move the text. In MaxThink we issue one command "M 45 B 5.6", which means move the current topic or group of ideas labeled 45 to the position in front of (Before) the sixth topic or group of ideas on level 5. Ideas can be moved around without positioning the cursor at all.

Beyond Word Processors

Thought organizers have the capability to store and maintain ideas and specifications for general model design as part of the top-down design methodology. Some Thought Organizers have the ability to store blocks of text and graphics by multiple keywords. This is similar to database functions of filing information as structured sets. One can then sort these into different sub-sets. They have easy to use navigational capabilities to support design iteration as the designer makes changes to all levels of the design hierarchy.

MaxThink as a Thought Organizer

The functional characteristics of MaxThink illustrate the power of thought organizers in designing simulation models. There are two classes of functions for changing the design structure of a simulation model. The first allows the designer to change the order of a list of topics or a group of text at a given level in the hierarchy. The Prioritize command rearranges a list of topics or ideas. The designer can specify the topic numbers in priority sequence and the list is not only reorganized, but also renumbered into the new sequence. The Randomize command performs the opposite by shuffling a list into a random order so that the designer can begin to develop a fresh viewpoint of the design. This can be used to prevent the continuance of a partiality developed 'mental set". The Sort command is built within MaxThink and allows topics to be sorted on any column and column width. A tabular structure can be imposed on a list of topics and sorted on any column as though the list were a small database. The Divide and Join commands are reciprocal. The Divide command splits up a single topic into multiple topics using criteria of lines, words, phrases, or paragraph splittings.

The second class of functions affects the hierarchy of topics. The Binsort Command allows the designer to create categories for topics and then to place the topics into the appropriate categories. This is helpful in model building when the designer discovers higher levels of organization to the design specifications. This supports the process of mental induction. The Fence command adds fences or boundaries to a list of topics or text that is already in correct order. The Categorize command can then convert the fences to topics and automatically subordinate the fenced topics under the associated fence. This supports the process of mental deduction or hierarchical subordination. The Levelize command is the reciprocal of the Categorize command. There are times in the development of the model when we wish to dissolve a part of the hierarchy into a linear list in order to refashion it.

Proof of the Pudding

The Author has found from personal experience that the time to design and implement simulation models can be cut to at least 1/3 when compared with traditional methods using pencil and paper, cut and paste, or even word processing. An additional advantage of thought organizers is that this same tool can be used for both simulation model design and implementation. After the design is finished, the computer programming of the model can be an extension of the design by being included as the lowest levels within the design hierarchy. This gives a continuity to the design and implementation processes and produces a coherent set of documentation ranging from the highest levels of model abstraction to the lowest level of computer code.

SUMMARY

Thought Organizers are powerful tools for the development of small-scale simulation models, when these models are being developed by individual designers. They aid in the construction and continual maintenance of hierarchical structures for the creation of simulation model specifications. They support well-accepted structured design techniques and go beyond them to support iterative design in which changes can be made to all hierarchical levels of the design during the creative design process.

REFERENCES

[11 Fuhs, F. Paul, "A Simulation Model for Calculating Session Time For Running a Simulation Model in a Shared Resource Environment", 12th Annual Conference of ABSEL, 1985.

[2] National Bureau of Standards, Software Development Tools, 1982, NBS Special Publication 500-88)

[3] Harrison, Allen F. and Robert M. Bramson, Styles of Thinking, (Anchor Press: N.Y., 1982).

[4] Filley, Alan C., Robert J. House, and Steven Kerr, Managerial Process and Organizational Behavior (Scott, Foresman, and Co.: Dallas Texas, 1976)

[5] Orr, Ken, "Managing the Software Crisis," Computerworld, July 15, 1985.

[6] Nicholas C. and Carolyn Mullins, "The Organization of Thought", PC Week, January *29,* 1985.