IMPROVING THE COMPUTER INTERFACE IN
ARCHITECTURAL EDUCATION

by

Ronald Fergle

Bachelor of Environmental Design
Miami University
Oxford, Ohio
1983

SUBMITTED TO THE DEPARTMENT OF ARCHITECTURE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE
DEGREE
MASTER OF SCIENCE IN ARCHITECTURE STUDIES AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1986

© Ronald Fergle 1986

The author hereby grants to M.I.T.
permission to reproduce and to distribute publicly copies
of this thesis document in whole or in part.

Signature of the Author

Ronald Fergle
Department of Architecture
May 15, 1986

Certified by

Harvey Bryan
Assistant Professor of Building Technology
Thesis Supervisor

Accepted by

Julian Beinart
Chairman
Departmental Committee on Graduate Students
ABSTRACT

The influence of the computer is increasing within the architectural profession. One aspect of this is the growing use of microcomputer programs in architectural education. Many of these programs have their roots in the engineering disciplines, and therefore their procedural methodology may not be compatible with the architectural design process. In addition to this, most of the programs used in the universities are originally designed for the professional practitioner, and may not be appropriate for the academic environment.

This discussion explores the characteristics of the architectural design process, and what issues programmers need to address when writing software for use by designers. More specifically, how should the programmer approach designing educational software, so that the computer becomes a more effective tool in enabling the student to develop heuristic knowledge about some aspect of architectural design.

The programming factors that influence the effectiveness of this type of educational software include: the appropriate use of graphics, flexible input/output sequences, procedural transparency of the program structure, and the iterative comparison of design options. These concepts are analyzed in a series of programming examples involving energy analysis and daylighting analysis.

Existing programs are critiqued, and suggestions for improvements are made. The use of processors to facilitate the testing and comparison of results are presented, as well as guidelines for additional developments using knowledge base overlays.
Donald and Evelyn Fergle – my parents. Without their financial and moral support, none of this would have been possible.

Harvey Bryan – For helping to clarify many of these ideas, and for providing suggestions based on a tremendous amount of knowledge and experience.

Fuller Moore and Ann Cline – for planting the seeds that grew into my interests involving daylighting and computers.

Sergio Vasquez – for helping to put this work in a pretty package.

And Chipper – for just being himself.

Thanks.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>CONCEPTUAL PROGRAMMING ISSUES</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Environment</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Architectual Design Process</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Production aid vs. Education tool</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>ACTUAL PROGRAMMING DEVELOPMENTS</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Past: MICROLITE 1.0 and PASSIVE 1-2-3</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Present: MICROLITE 1-2-3</td>
<td>42</td>
</tr>
<tr>
<td>4</td>
<td>PROPOSED PROGRAMMING STRUCTURE</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Educational Context</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>INTERLITE Program</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>CONCLUSIONS</td>
<td>60</td>
</tr>
<tr>
<td>Appendix</td>
<td>EXCERPTS OF MICROLITE 1-2-3 CODE</td>
<td>66</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>73</td>
</tr>
</tbody>
</table>
"Are you from MIT and cannot read, or from Harvard and cannot count?"

Broadway Supermarket Cashierperson.
THE ISSUES

Computer analysis programs written to aid architects in their professional design practices have increasingly been used in design schools to supplement the education of the students. Regardless of how well these programs work in the professional environment, how effective are they in the academic environment? Do they hinder or encourage a student's comprehension of the concepts underlying the issues involved in architectural design? Are they simply black boxes that return a value based on a particular set of circumstances, or are they transparent enough to be able to help the student develop an intuitive feeling of the effects that each component of the situation contributes to the final result?

While there is a place in architectural education for programs written specifically to aid the professional, there needs to be a greater effort to write programs designed with the purpose of supporting the learning process of students. The professional packages generally do not adequately address the methods of inquiry used to formulate an understanding of the conceptual relationships of a particular problem. The educational programs should contain increased flexibility and ease of information transfer between the user and the computer; more emphasis on comparing the effects of incremental relationships of particular design elements; and more flexible structuring of the programs so that different users can approach the same problem from various, individualized methods.

While efforts to improve and incorporate computer programs as educational tools is relatively new in architecture circles, work has been progressing well in this area in other disciplines. Computer aided instruction is emerging as a field
with tremendous potential in areas such as troubleshooting and repair of mechanical equipment, as well as simulating conditions in situations such as nuclear reactor management and steam plant control.

An example of one of these systems is STEAMER, developed by the Navy Research Personnel Development Center of San Diego, California and Bolt, Beranek & Newman, Inc. of Cambridge, Massachusetts. It is designed to teach users how to manage and control complicated steam propulsion plants, like those found on naval vessels. It utilizes two display terminals, a keyboard and a mouse as I/O devices. One terminal displays a general overview of the major components in the steam plant. Each component and the flow relationships between them are shown graphically. Using the mouse, the user is able to point to and select one of the components and give the command to "expand", and the screen is replaced by a new diagram showing the relationships in that sub-system. The other terminal shows more specific numeric as well as visual plant information. The student may alter states or change the conditions of simulated dials and meters as if he were managing an actual plant. He can then observe the resulting effects on the diagrams.

While STEAMER is not an architectural system, it demonstrates some worthwhile features that architectural packages might implement. First, it makes ample use of color and graphical displays to relate information about the plant. In this manner, relationships can be perceived more clearly than having the user only attempt to interpret series of numbers. Second, the student can use a mouse to manipulate the system. While access to a keyboard is still essential, this increased flexibility gives the user more freedom to act on a thought. Third, the system performs in real time. The user can make changes and watch
how the different components respond. Fourth, the program allows him to save the sequential interactions. He can come back at a later date to try some other responses at a particular junction and observe the new results. Fifth, and probably most important, is the approach of the system to be essentially a "dictionary". It does not try to "dictate policy" by using a standard, non-flexible format to tell the user what order to change valves and by how much. The student is able to experiment in whatever sequence interests him, thereby personalizing the instruction.

Some of these features are made possible by specific elements of the computer system. Contributing to the real time processing is the speed and information handling capabilities of the hardware and the underlying numerical relationships of the steam plant. The graphic interaction is facilitated by the capabilities of the INTERLISP programming language. Yet, the concepts to strive for are there. Additional considerations need to be addressed when moving to an architectural context, but STEAMER is still an example of the potential improvements in a user/program interface.

The community that writes software for use by architects and other design professionals must become more cogniscent of how their programs fit into the overall process of design. There needs to be a greater sensitivity by the programmer to accommodate the interactive, open ended approach to problem solving and knowledge acquisition used by designers.

Programs should not be written simply to produce numeric values based on certain data configurations, but should be structured to become part of the design process themselves. They need to become a tool as manipulatative and user directed as a pencil. In that way, the user may explore conceptual ideas.
and relationships in a manner that is most comfortable to him, and therefore more productive. Analysis programs used by students should also be transparent, so that the user gains an understanding of the foundational concepts and relationships that effect the numerical values returned by the program. Only in that way, may these packages be truly called successful educational tools.

This discussion will examine the issues involving the interface of computer analysis programs and the architectural design process. A priority will be to explore the concepts that will permit these programs to become more appropriate tools for use in education. The specific context is in microcomputer programming dealing with interior daylighting and energy analysis. The issues raised are not based so much on reactions to "hard data" from performance testing, but on personal observations and experience gained from programming this type of software.
Chapter 2

CONCEPTUAL PROGRAMMING ISSUES

"You can't think seriously about thinking without thinking about thinking about something."

Seymour Papert
THE LEARNING ENVIRONMENT

What are the aspects of the educational context and learning process that computer programs have to fit into? To write programs with the purpose of improving and developing the general surface knowledge of the user, the programmer needs to be particularly aware of the more ambiguous learning environment. When the computer program is initially being developed and conceptualized, certain erudite issues need to be taken into consideration because the internal structure of the program must facilitate these concepts. The program's foundation must be laid correctly for the resultant user interface to be as effective as possible.

Heuristics are rules-of-thumb simplifications of the ideas and relationships involved in a particular issue. (1) They are personal understandings of the general concepts driving a relationship. The way each person internalizes and builds up the knowledge to develop these rules of thumb is unique. For architects, visual representation is a significant part of this process.

A goal of using these programs should be for the student to develop heuristic knowledge about some aspect of the architectural design process. When a person uses an analysis program he is examining a building configuration at a specific point in the design. He gets results based on the configuration at that particular instant. The objective is then to relate the knowledge derived about that particular situation to other similar cases, and thus to put it in a framing context. That way when designing later spatial configurations the student will be able to recall the earlier conclusions and make a more educated proposal.
In the case of daylighting, that would involve understanding the relationships of surface reflections, spatial configurations, window placements, and orientation, to name a few. How do all these variables act together and what are the effects? What will the general light levels be, what will the gradient relationships look like, and will there be glare? An architect will develop a feel for the answers to these questions through years of professional practice. He will form intuitive understandings or intelligent guesses to these issues by going through the process of designing, building, and observing the resulting effects. However, a student could be able to develop some understanding of these concepts while still in school if the computer programs dealing with lighting are designed appropriately.

One of the main issues is how the user approaches the problem. His attitude toward it. Seymour Papert in Mindstorms recounts a situation where a particular learning method was not appropriate for the student. (2) In this case, an elementary school student was not able to adequately understand the ideas and relationships behind multiplication tables. He was therefore doing quite poorly in class. The student stated he was trying to learn the tables by first "letting his mind go completely blank", and then repeating the material over and over. (3) The student had poor retention because he wasn't comprehending the concepts involved in the multiplication process. He was only concentrating on the superficial mental imagery, trying to learn through a non participatory, wrote-memorization process. As Papert says, "[he] failed to learn because he forced himself out of any relationship to the material - or rather, he adopted the worst relationship, dissociation, as a strategy for learning." (4)

I believe there is a similarity between the way the child approached learning multiplication tables and the way many of the existing computer analysis
programs are used in education. A person runs an analysis on a particular building configuration and obtains a result. He may then make a change or two and run the analysis again. But he doesn't really LEARN anything. He may perceive some identities based on a certain configuration, but fails to fully comprehend how all the pieces really fit together and interact. This is because he is dissociated from the knowledge. He is really just going through the motions of analysis.

The user is removed from the "nitty-gritty" of the mathematical process and calculations because the program is doing it all for him. He doesn't see the equations and run through them himself. He simply puts a value in, gets a value out. This imposed black box approach is like the child's dissociation. Because there is no direct link or connection to the mathematical relationships of how the variables effect the results, the learning process in using these programs is essentially just another form of wrote memorization.

In the case of the elementary school child, Papert succeeds in getting him to achieve tremendous improvement in understanding the concepts of multiplication through the use of the LOGO programming language. LOGO utilizes a very user friendly and participatory format in that graphical images can be created by combining a series of simple commands together. (5) It is that these commands or instructions are actually an application of vector geometry that they succeed. Because the user is able to imagine how he would tell himself to draw something, he is able to instruct the computer to do the same. (6) The key is that the student can form a relationship to the information because he is able to associate it to something he already knows.
Likewise, programs dealing with daylighting should allow the user to interact with them in such a manner that the student can question and inquire about the effects of changing different elements, and view the relationships in a form that allows him to perceive the trends and associate them with something he already knows. This means that the program output must be flexible and conducive to manipulation by different users.

If graphs are appropriate for the type of information to be conveyed and the student is comfortable with them, this must be an option that he can choose. If sections or axonometrics represent the data better, then these must be available also. If simple numeric values are desired, they must be presented. It comes down to the user having the choice to decide in what form he wants the results displayed. If the program only provides one format, a numeric printout for example, then chances are that most of the relationships will not be perceived and retained by the user.

The programs also need to allow the user to "see inside" and remove some of the black box characteristics that so many have. If the relationship of one variable to another is hidden in some algorithm buried deep within the code, and the program does not have a flexible, iterative format, then the user will probably never fully perceive the connection between the factors. The student must be allowed to deduce these relationships by using the program and experimenting with various configurations in such a manner that is both interesting and personal.

The process of learning without being taught is referred to as Piagetian learning. (7) It is a more open approach to learning because there is not an emphasis on absolute "right or wrong", "yes or no", but instead an emphasis on "by how
much", "can it be fixed?" This attitude and the whole issue of being able to explore and question different alternatives in what ever manner and order a person desires, is a key concept for the programmer to keep in mind. This flexibility permits the student to create personal associations of the concepts involved in the problem.

It is in this way that a student using a daylighting program could experiment with ceiling height and window placement because at that time he is quite interested in the development of Gothic cathedrals. Are the regional differences from Southern Europe to Northern Europe based on a response to the changing light characteristics of the latitude and climate? For this student, the program becomes a tool to help him develop a personal understanding of some of the design generators of a particular aspect of architecture that he is currently quite interested in. If the program is inflexible in its presentation and format, he won't be able to do this. He is also able to relate general daylighting principles with another area of architecture that he is already familiar with.

Papert suggests that learning something is easy if a person can associate it to some model already existing in his mind, and the more models he has available to reference, the better. (8) This is similar to the theories of how human memory functions. Information is segregated into chunks and the person references the chunk with the appropriate information when needed. Chunks can receive more and more detail as time progresses and new associations are made. (9)

The issue in programming educational software is to present the information in such a manner that a user can identify it and store it in whatever manner is best for him. Each person should be able to create the conscious and
subconscious relationships and associations that most effectively aid him in recalling the information. However, it is initially a matter of the user being able to perceive the information in the first place. This is why the flexibility of output and ability to question the program in whatever manner desired is so important.

As a person adds new elements to a particular concept in memory, his collective associations about that concept grow. As there is no apparent limit to the level of detail and amount of information contained in these chunks (10), the more experiences the designer can add to his repertoire, the greater his design palate will become.

If the computer program is so rigidly structured that there is only one way to do something or address some issue, then many users will not be able to form an adequate, personal relationship to the material. Looking at the same problem from a number of different viewpoints is advantageous when trying to understand the situation better. In addition, because everyone is an individual and looks at things from a slightly different perspective, that personal fit with the program is even harder to achieve. However, if a programmer conscientiously tries to permit multiple access to the inherent concepts within the program, then more people will be able to attain an understanding of the issues.

The programmer needs to realize that any one particular problem solving methodology is not necessarily better than another. However, some may be more appropriate. This is why the programmer needs to understand the way architects approach a design problem so that the program can more accurately accommodate the process. This understanding is important because much of the program – user interface is dependent on the assumptions and coding routines
initially used by the programmer. To maximize the learning effects of the program, the procedural flow of control within it needs to "be in tune" with the general way designers think. It needs to be a sympathetic sequence.
THE ARCHITECTURAL DESIGN PROCESS

While architects don't all necessarily think the same or approach designing with the same conceptual priorities, the general framework of the method by which they develop their particular concepts and proposals is similar. It is an interactive, if–then type reasoning where new information is constantly used to update the scenario. It is a process of experimentation and exploration, trying new ideas out while always maintaining a dialogue with past experiences.

The design process is unusual in that it is a process of top–down design as well as bottom–up. From general concepts to more specific actions, and from individual details to an overall consistancy. How the designer balances the two approaches at the same time is a unique aspect of the method. He must constantly oscillate back and forth in order to achieve a coherent proposal.

To do this, he goes through what Donald Schon calls "reflection in action". (11) The designer interprets the information of a particular situation that he has in front of him, and on the basis of his perceptions and personal characteristics, performs an if–then experiment in which he receives new information about the situation as well as a simultaneous modification of that situation. He is able to reframe his awareness of the problem, and in so doing, slightly changes the problem.

The designer's perception of the problem continually changes as he goes through series after series of question and answer sessions. To reframe his understanding of the issues, he must constantly evaluate his proposed modifications to the scheme. These changes are interpreted based on their relation to general
design requirements, their consistancy to earlier design decisions, and their resulting effects to the design situation. (12)

The designer needs to be attentive to the changing characteristics of the design scenario as the process progresses. It is in that way that he can perceive new possibilities for action. The architect cannot know initially where a series of proposals and evaluations will lead. They may be worthwhile decisions, or they may end up having negative effects. If the sequence has an undesirable outcome, then the designer backs up to a previous point in the design and reflects on the proposals that were just tried. He uses the information to more accurately frame his next theory. This is not a situation to be avoided, because the only way to become more confident about an apparent understanding of something is to question it. In this case, the designer is able to repeatedly fine-tune his perception of the situation.

As the designer moves through this process, his experiences and knowledge gained from previous design sessions and construction results are recalled for reference. The new proposals are compared to these earlier cases, and the factors that were pertinent in these models are used to help evaluate and judge the current situation. As the architect becomes more experienced, his repertoire of models will increase. This allows him to make more sophisticated decisions. He will be able to recognize a similar configuration from an earlier design and recall what the consequences of that design decision are. This will add to his ability to make the appropriate design move in the current problem. This immediate experience will in turn become another model to use in future sessions.
The designer is not just an observer in this method, he is part of the process. He judges, evaluates, and proposes based on his personal perceptions and philosophical beliefs. This is how designs become unique and in many cases personal statements by the architect. However, the design process is not simply subjective. It is a combination of both objective and subjective reasoning. The subjective quality gives architecture its flair and expressive creativity. The objective aspect gives architecture its functionality.

The subjective design considerations are tempered by the objective evaluation of their feasibility. To be able to successfully criticize a proposal, the designer needs to be well versed in the physical characteristics of construction. He needs to have a handle on the mathematical relationships or the specific properties of the actual design feature. This is why an architect must be aware of the physical relationships of such issues as structural integrity, material properties and energy use, to name a few. The issue now becomes how to educate the architecture student about these properties in the most effective manner possible.

There are many of these such factors that architects have to balance when designing. The numerous relationships of the internal elements of the design are extraordinarily complex. One of the ways to develop the ability to handle all of this data is to concentrate on one particular aspect at a time. Attain proficiency in a certain area and then move on to another. Eventually the designer is able to deal with the interactions between these areas on a progressively more sophisticated level.

The objects being created, buildings, are composed of a series of identities ranging from three dimensional spatial features to philosophical meanings and
statements. Designing buildings means juggling a tremendous amount of information and concepts in the designer's mind at the same time. Models and drawings become simply methods to help the architect imagine how the resulting space will exist.

The architectural design process is an example of cognitive flexibility. The process involves the mental manipulation of three-dimensional objects and spaces. Being able to think in three-dimensions is critical for the designer, because the finished product is an actual physical structure as well as a sequence of spaces. Visual imagery and conceptual transparency are important elements of this ability.

The transparency is a major design component in terms of the graphical representation of the underlying concepts in the design phase, as well as the physical representation of the underlying conceptual ideas in the built form. As architects design, lines on the paper take on multiple meanings. In addition, the architect is able to look past these physical marks to the conceptual properties and mental images beyond. Various types of drawings and physical models are simply efforts to help the architect imagine what the space is going to be like and what properties it is going to have. The old adage that a picture is worth a thousand words is appropriate here in that the cognitive images that these medium transmit are what is important, not the actual physical qualities of the representations.

Architects think visually, so software that must fit into the design process must be able to communicate information visually as well. More than simple numerical values are necessary. Whether this means graphs or various types of drawings and spatial representation is dependent on the information to transmit.
and choice of the designer. The selection of graphs to represent trends, graphs to display quantities, drawings in section, or drawings in axonometric, should depend on the information to convey. (13)

The programs used by architects to analyze a spatial configuration for some particular property, such as lighting levels or energy use, are quite often written by mechanical engineers. This is because the in-roads of the computer into the architectural profession has been relatively slow. The automotive and mechanical engineering fields have been utilizing the computer for years. But architectural design and mechanical engineering are involved in different contexts. Programs designed for one may not be appropriate for the other. Each field has its specific issues and design goals, and therefore its own logic process in how to address them.

This "logic methodology" is one of the major characteristics that program designers need to address. In architecture, it is a very free-flowing, give and take approach. Numerical values are not always present to quantify a concept. Many times a range, or "feel" is sufficient to represent an idea. Analysis programs generally need correct, precise numbers to be entered in order to be able to return accurate values. Otherwise, it's probably GIGO (garbage in, garbage out). Because much of design is conceptual and hazy, at least in the early stages, the engineering based programs that place a lot of emphasis on numerical precision cannot really be used efficiently.

The procedural interface as well as the physical interface are contributing factors to the limitability of many of these programs. How these programs are structured to accommodate the logical, or illogical as the case may be, architectural thought process is a relevant concern. It is not simply an issue of
using mice, spiffier graphics, and various display formats, but rather in what way are these additional features implemented.

There must be a flexibility of the question and answer interface. A program cannot simply prompt the user for pieces of data, line after line, and then spit back a number or a table of numbers and expect to get much use by the designer. The user should be able to concentrate on a particular aspect of the design and then be prompted only for the relevant information.

When designing, the architect or architecture student should be able to use the computer program easily and quickly, so as not to lose his train of thought. Designing is a dynamic, spontaneous action, and if the designer has to stop everything to set-up and run an analysis program, then that program will not be used often, if at all.

There should be as little repetitious input of data as possible. Design is like a stream-of-consciousness process in that ideas will suddenly occur to the designer without his expecting them. Schön describes how one particular design move can lead to unexpected results and new, unforeseen possibilities. (14) When an architect is pursuing some train of thought, he can’t be restricted or held back by the program.

The data entry interface is one of the critical areas for the program designer to concentrate on. If the input data that is required is already existing on some other file, then the user shouldn’t be required to enter it again. Even if it is in various forms or different units, such as metric and English measurements. The program, or one that the analysis program can link up
with, should be the one to "do the work", to make the conversions and enter
the data.

If the user is designing with the aid of a graphic editor, the analysis program
should be able to extract the necessary information for it's analysis. The
designer should be able to have the flexibility to pursue his ideas in whatever
method or form is most convenient to him at that time.

It is important that the role of the computer program should be a tool. It
should be a design aid, not a policy dictator. The program should support the
design process and the unique qualities it possesses. It shouldn't noticeably alter
the way the architect approaches the design problem other than becoming
another tool at his disposal.

The architect or student shouldn't have to concentrate on any particular syntax
in order to communicate with these programs. The information transfer should
be natural and effortless. The process of transporting the data from the user
to the program should be transparent. The user shouldn't worry about how to
enter the data, he should simply be able to do it. One example of this type
of process is reading and writing. A person is able to "see through" the words
on the page to the meanings behind them (15), and generally the mechanical
process of writing words has also become an unconscious act.

The objective of the programmer shouldn't be how to get the user to talk the
computer's language, but vice versa. One of the ways to do this is to provide
many options for the user to use to enter the data. This could be mice, tablet,
keyboard, existing files. The language structure should also be natural.
ComputerVision's GBD (General Building Design) package requires a noun-verb
structure for command input. It is very bothersome to use. The user should have the option to enter data and commands in noun-verb or verb-noun if he wants. It is certainly more trouble for the programmer to plan for this, but is a feature that is essential.

For computer programs to support the architectural design process and the methods architecture students use to acquire new knowledge, the programmer needs to take into consideration these general characteristic of the design process that have been presented. While the architectural process is unique, it cannot be so specifically defined and identified as to say exactly how every designer actually thinks. It is just that the characteristics presented are relatively common among designers, and they form a bases for the development of individual design philosophies.

Generally all computer programs used in architectural design will benefit by acknowledging these procedural design concepts. However, the reasons and goals for which individual users run the program vary. Therefore the programmer needs to be aware of these distinctions and plan accordingly. This will enable each program to respond to its specific context in the most appropriate manner.
PRODUCTION AID VS. EDUCATION TOOL

One of the major distinctions in the context of how programs are used in the architectural environment is whether they are used by an architect in an office or by a student in a university. The individual goals for each user are slightly different. Yet, the same programs are used in both situations rather indiscriminately.

For the most part, the majority of the software used in architectural education is initially designed for the practicing architect. Reasons for this range from basic economic returns for the programmers to the fact that there is simply a tremendous need for production aids in the work environment. It is then quite often through liberal licensing agreements that the schools receive these "production" programs to use in their curriculum.

One of the more noticeable characteristics of production programs is the emphasis on returning values. There is also a lot of importance placed on the accuracy of the values they calculate. The goal is generally not to educate the user as much as it is to return numerical values that the user can have confidence in. They are geared more to provide QUANTITATIVE information as opposed to developing QUALITATIVE understandings. There is beginning to be work on programs that return qualitative interpretations based on trends (16), but the majority of the existing software isn't designed like this.

These programs are usually used to analyze a specific situation further along in the developmental stage, after the initial design decisions have already been made. It is more of a fine-tuning process of the concepts already developed.
Essentially the programs are used to confirm or reinforce the user's existing knowledge of the situation instead of helping him develop new understandings.

When it comes to analysis programs, the driving force within them are the mathematical algorithms. In most programs, the mathematical methods used to derive the results are not displayed, nor is there a concentrated effort to involve the user in the calculation process. The user really has no contact with the analysis routines because they are hidden from him, buried deep within the code. He is not exposed to the inherent assumptions and biases of the algorithms. Consequently, the programs are black boxes.

When production programs are black boxes, the user is generally not able to internalize the cause and effect relationships of the contributing factors that determine the final numerical outcome. This is because it is not the major objective of the program. However, in the academic environment where there is allegedly an attempt to distill knowledge, this should be a priority.

When a production program is used in an educational setting, one approach is to run a lot of iterations. The student changes the value of one variable and runs the program again. He then compares the results to see what effect the change has on the final outcome. In this way the student can begin to see what sort of cause and effect trends develop.

When the program takes a long time to execute, this process becomes very tedious and boring. If the data entry sequence is long or awkward to use, this compounds the effects. Furthermore, if the program doesn't even support this method because comparison of different runs is not directly possible, the user
may really not learn very much. He is still quite removed from the actual analysis process.

Another current method of using these analysis programs in education is to combine the computer analysis with long-hand calculations of the same problem. This way the student has to actually perform the mathematics and think about the analysis process first-hand. But interpreting results is still reduced to comparing a few numerical answers and the process can take a long time. This is a learning approach used often, but it really is not utilizing the inherent capabilities of the computer very well.

Production programs are also generally slow to use and awkward to implement. The typical production program requires a lot of data to be entered because accuracy is a major priority. This means that the input sequences can become quite involved, and the time it takes the program to calculate the results can become quite lengthy. When programs have to handle an increasing number of variables and mathematical calculations, the time it takes to run them becomes longer. When the buildings or the spaces to be analyzed are complex, the whole process can become a major undertaking.

These programs are really not designed to be used as educational tools. The input information required is usually so detailed and specific that trying to simply concentrate on an underlying conceptual issue is hard. The user really needs to have an existing design before him in order to utilize these packages.

One way to improve the use of these programs for learning purposes is to create the most generic and simple spatial configuration possible to analyze. Many of the variables would then need to be set to a value with the most
minimal effects on the outcome in order to create a comprehensible "base case". But a person initially unfamiliar with the relationships may not realize what values are appropriate default values, especially as different base cases change. In addition, this is in essence a "hack", tricking something into doing something else for which it isn't designed to do.

If this process is used, then the code is not being maximized for the kind of situations that it could be. The ability to handle complex configurations is being wasted. A lot of the data required would also be unnecessary for the general theory testing used by the beginner, and so it would really be wasting some of his time also. The way these programs are structured now, he would not know which of the variables are trivial. He could waste effort exploring dead-ends and not realize it. He could get lost examining the specifics of a tree and not see the whole forest.

The non-efficient use of the program is one negative factor in using production programs in a method that they are not designed for. But a more potentially dangerous scenario exists if a situation is forced into the program format for which it is not appropriate so that some type of analysis can still be run on it.

This is another hack, but on the opposite side of the coin. The user may disregard features of the design that are not accounted for in the program. Therefore, these effects will be totally left out of the final results. The conclusion will be misleading. For the knowledgeable designer, this could have the effect of allowing him to manipulate the results (17), and for the student, the proper understanding of the relationships will not materialize.
In the case of energy analysis and material properties, or daylighting analysis and window fenestration, information left out or modified can have a pronounced effect on the results. Economic payback projections may be altered or mechanical systems could be sized incorrectly. "Playing with the numbers until they say what you want them to say", is a common phrase that describes this process.

If programs are to be used in education, the interactive if-then questioning format of the design process should be the element to concentrate on when writing them. Because acquiring the heuristics of the various issues that effect design involve the process of hypothesize-test-evaluate to such a degree, the programs should target this approach. That way they will be more responsive to the learning sequences, and much more efficient as educational tools. As it is now, they're generally not designed with this as a high priority, so when used in educational settings they are not as beneficial as they could be.

As more architects or people with an architecture background become involved in programming educational software, these problems will diminish. However, many of the programmers currently writing architectural software are concentrating on packages designed for the work environment. This is mostly a matter of economics. The programmer is able to charge a higher price for production packages, and there are many practicing architects willing to buy them.

The academic environment isn't a controlling factor in the market. It usually receives the software as donations or for a minimal fee. Probably because the programmers feel that if a student becomes acquainted with their software while in school, they'll more likely use it when they graduate. They are more concerned with educating the student about the software packages, and therefore
eventual sales, than they are with specifically teaching the student about the
design concepts.

So, the programs used in education are generally not designed specifically to
educate the student. The features of these production programs are not
necessarily the most supportive of the academic issues, but their use within the
universities in never the less quite extensive. For programs that more
appropriately address the academic concerns of architectural education, there
should be a greater effort on the part of programmers in the universities
themselves to write the software. (18) This process is slowly taking effect, and
hopefully the contributions will soon become an integral part of architectural
education.
"In the prelude to an Architect-machine dialogue
the solidarity of the alliance will rely on
the ease of communication, the ability to
ventilate one's concepts in a natural vernacular,
and on the presence of modes of communication
responsible to the discipline at hand"

Nicholas Negroponte
PAST PROGRAMS

There are a few architectural programs currently existing that attempt to cross over between the production and the academic environments. The process of continuing to develop these programs and the theories driving them is an on-going effort. This paper is just one aspect of this concern. While this facet of "computer-aided-instruction" in architecture is still in its infancy, the potential benefits are encouraging.

Many of these programming efforts were written within an academic environment to begin with. This is probably one reason why there is an attempt to address the factor of learning when using them. By looking at some of these programs, the various trends that are emerging from one effort to another can be identified.

Many of the deficient aspects of the early attempts at daylighting and energy analysis software reflect the nascency of the whole computer industry at the time when they were written. The order of program development went from hand-held calculators, to batch processing with large main-frame computers, to long-distance telephone communication with time sharing programs, to individual microcomputer packages. (19) FORTRAN was the language of choice for many of the mainframe efforts because these were generally coded by engineers that were already familiar with computer programming. BASIC was used for the microcomputer versions because they were usually written by architects that had little formal programming experience.

BASIC is a relatively easy language for beginners to program in because there are not many syntactical or structural restrictions imposed on them. However,
these characteristics lead to many of the problems found in these early programming efforts. This unstructured format tends to encourage "spaghetti code". This is where the flow of control jumps all around within the program. To make major changes and updates to an existing code is virtually impossible.

The full development of an effective information transfer interface is also inhibited. The coding acrobatics a programmer has to go through to create an interesting screen interface is quite a chore. The way variables are used and defined internally also causes problems. A result of these limitations is that the programs take a fairly long time to execute. When an interpreted version of BASIC is used, it becomes even worse.

The C language is a more recent development in programming languages. It has a more structured format. Variables must have their types declared and they must be treated in a certain manner when used with functions. It is also a more modular language, and programs written in it lend themselves to easy updates and alterations. C programs can also be transported from one machine to another more readily. (20)

One of the early microcomputer packages that deals with daylighting is MICROLITE 1.0, written by David Krinkle in 1983. (21) Over–all it is a well written program considering the language and machine limitations. It is coded in BASIC and is designed for a personal computer with relatively little memory. This combination leads to slow and awkward handling of the information by the program. It is also a "universal" program. All of the data entry, analysis calculations, and screen graphic displays are handled within it's own immediate code.
The basis format of the program is to input the information about a space to be analyzed through a series of question and answer menus. All of the information is entered through a keyboard. The data is structured in segments. One segment contains the specific information about the room configuration and reflective properties of it's components. A second segment has the information about the windows, and a third one contains the site data and orientation. The user can piece together different segments to create the final analysis configuration.

This general structure facilitates the comparison of different over-all configurations. Users can call up and modify any previously created segment. The earlier entries become the default values. The user can change as many individual entries of the previous segment that he wants to in order to compose a new segment.

To run a new analysis, the user can then pick this new segment as one of the components or choose an entirely different one. However, the option of changing whole segments at once may in the long run be undesirable. This is because the user is switching entire units which contain a lot of information. Therefore he is actually comparing the effects of the altering of many different variables. Is it possible for the user to comprehend the underlying relationships when so many factors are changing at the same time, and should the programmer encourage this method of analysis?

The lack of memory within the computer dictated the policy of creating individual segment files. The machines weren't large enough to handle all of the information at once, so it had to be broken down into smaller units. When the time came, the pertinent data would then be analyzed. While this
basic structure was a requirement and the programmer had to adhere to it, the issue of providing so many options to the user should be discussed.

It may be desirable to give the user a lot of options in certain cases, and fewer in others. It really comes down to what is appropriate for that particular aspect of an analysis. When entering the data into the program, providing a wider range of choices is better. In the case of changing entire blocks of information, the programmer has to be careful. If the option of changing whole segments at once is supported, then all of the information within a segment should be closely related. Hopefully this will create an identifiable block of information. One that is easier to comprehend.

The user needs to be able to perceive what changes have what effect on the outcome. If changing entire units results in the user losing track of the concepts because there are a variety of relationships contained within each unit, then this option is deleterious to the learning process. Instead, let the user experiment with the specific, individual changes so that he can perceive what the direct effects are.

In the case of MICROLITE 1.0, the segments contain data on different types of windows and different site locations and characteristics. These units are fairly identifiable, so it is probably not detrimental to permit this type of comparison. However, when assembling the analysis configuration, the ability to simply switch one segment label for another without actually thinking of what information composes that segment is still a possibility.

The approach of MICROLITE 1.0 in displaying the calculated lighting levels graphically is quite successful. The user is able to choose from a number of
different visual methods to view the results. He has the option to pick numerical representations in plan form, various sections, or axonometric projections. Although the data for every point in the room configuration has to be calculated no matter what option is chosen, the significant feature is that the user is able to decide in what form he wants to view the results.

One of the major drawbacks to the program is the time of calculation. Part of this is due to the number of points to be calculated (twentyfive), the internal structure of the BASIC code, and the fact that the program calculates all of the geometries and graphic coordinates for the display itself. Another significant reason is that as the program calculates the level of total illumination for each analysis point, it writes every contributing sub-value on the screen. The time it takes the computer to access the screen to display each value is a tremendous waste of time. The user simply doesn't gain anything watching these numbers scroll by anyway.

An additional drawback is that results from different runs can't be compared to each other directly. The user is able to calculate all of the results first and then later on load in the values to be viewed. But this method requires the user to remember the shapes and characteristics of the previous runs when flipping from one graph to another. Direct comparison is not an option.

Another program written that begins to bridge the gap between production and educational software is PASSIVE 1-2-3, written by Steven Lotz in 1985. (22) The program deals with energy analysis of small, 4 zoned, office buildings. The package contains two major parts. The first is a SOLPAS SLR type analysis based on a correlation approach. The second is a more detailed CALPAS analysis based on dynamic hourly weather data.
PASSIVE 1-2-3 makes an attempt to become more flexible in the data entry sequence. The SOLPAS portion can have information entered from two sources. One is the standard keyboard-menu method, and the other is information transfer from a file created using another program. The user is able to design a basic layout using a graphic editor, such as the AutoCAD package from Autodesk, and tag elements of the design with attribute values. This design information can then be used by the AutoCAD program to create a DXF file. This DXF file can then be sent to the SOLPAS section of the analysis.

SOLPAS is a quicker, "rough and dirty" calculation that allows the user to test a number of ideas relatively fast. When the user becomes more certain about the changes in the design that he would like to compare, the pertinent data can be sent to the CALPAS section of the package. CALPAS is a longer, more detailed analysis, so the user wouldn't want to use it if he wasn't really sure about his design options. (23)

The data transfer, or cascading of information from an AutoCAD design to a SOLPAS analysis to a CALPAS analysis, enables the user to avoid entering the same information twice. The only questions asked are ones for new, more specific information. This is because each section becomes slightly more precise in its calculations.

The CALPAS segment also has a batch processing routine. This is because this section of the analysis takes quite a bit of time. The calculations take from one half hour to possibly two hours depending on the machine used and the complexity of the space being analyzed. The user is able to enter four separate configurations to be analyzed and then let the program run overnight. In the
morning the user can view the results and compare the different design options.

The results can be displayed graphically using LOTUS 1–2–3. The PASSIVE 1–2–3 program contains routines that allow it to pass information to the LOTUS package for viewing. The display options are bar and line graphs.

Because the user can compare results of different runs next to each other with these graphs, he can begin to get a feel for what his changes are doing to the over-all building performance. These displays represent the final collective results, so when used by the student strictly as a learning tool, he would experiment by changing only one variable at a time. By proceeding through a series of runs and observing the results, it would be possible to internalize some of the cause and effect relationships.

The comparison of MICROLITE 1.0 and PASSIVE 1–2–3 display some of the major trends that are developing in this particular programming arena. One trend has to do with the linking of independant packages into an integrated system. This is evident in the change from MICROLITE 1.0's universal, do it all approach, to PASSIVE 1–2–3's ability to communicate with AutoCAD and LOTUS 1–2–3.

This linking enables the programmer to avoid having to "reinvent the wheel". Instead of having to program all of the graphic input routines for instance, the program can just be set-up to receive the information from some other program that is already focused on graphic editing. This helps free-up the programmer to permit him to concentrate on his particular analysis code.
A more flexible interface is also provided by this linking. Users are able to utilize programs they are already comfortable with, such as AutoCAD and LOTUS. As long as the filter programs are written to translate the data back and forth between the different programs, the possibility of expanding the system is limitless.

The next step in the trend is to provide a universal data base that can receive information from a various number of sources, such as AutoCAD, Microcad, and VersaCAD, among others. Then the specific application program can access the appropriate data that it needs. A further possibility is for two-way communication. This allows the application program to send information back to one of the sources for updates. This can either be direct or through an intermediary data base.

Another trend is the increased emphasis on iterative comparisons of design options. This involves previous file retrieval, hierarchical menu structure, and in some cases batch processing. This iterative process facilitates the interactive framework that is a characteristic of architecture and of the learning process. Reflection-in-action is encouraged by comparing and interpreting these different scenarios.

The improved graphics interface is a major contributor to this comparison ability. No longer are "dry numbers" the only way to compare results. Through the use of visual representation of the numeric quantities, and linking with programs that are tailored for graphical displays, the effectiveness of parametric comparisons is greatly improved. The user is able to achieve a better understanding of the relationships and quantities involved in the situation.
The factors that comprise these trends need to be addressed by the programmer at the coding level. The internal structure of a program needs to be supportive in order for these external links and iterative comparisons to be effective. Much of this is accomplished through modularity. The use of the C programming language is part of this, and the increase in its popularity among programmers parallels this over-all trend toward module coding.

These procedural trends are dynamic in that they are continually responding to new hardware developments. The conceptual goals and how to achieve them also change slightly based on knowledge gained from past programming experiences. This progress has been steady, and as programs are written and critiqued, the next efforts will become that much better.
CURRENT PROGRAMS

One of the programs that attempts to continue to improve on the trends exhibited in the MICROLITE 1.0 and PASSIVE 1–2–3 programs is MICROLITE 1–2–3, which is currently being written by myself. The program is essentially a repackaging of the original MICROLITE 1.0 algorithms in a *NEW AND IMPROVED* user interface. A concentrated effort has been made to try to reduce the black box characteristics of this type of analysis program.

The package utilizes information transfer from other computer programs for data entry and result display. In addition, processors that are able to set up batch runs to study the effects of incrementing certain variables are provided. The user is also able to graph the effectiveness of one type of incremental variable change in comparison to another, separate variable.

The program is able to take input data from AutoCAD via CRL, a data base designed specifically with graphic relationships in mind. (24) The data that can potentially be transmitted this way consists of room dimensions, window placements, ceiling, wall and floor reflectances, and glazing transmissions. This is achieved through the use of the attribute feature of AutoCAD, and its DXF file output.

A translator or filter program then extracts the pertinent information from the DXF file and places it in the CRL data base. MICROLITE 1–2–3 questions the user if information was placed in the CRL file this way. If it was, the program extracts it from CRL and places these values in the ABC, or Active Building Characteristics data structure within the MICROLITE program. These values have now been entered, and the user need not repeat the process.
Additional data about site characteristics, orientation and climatic conditions are then asked the user through a sequence of menus.

After a building description file is created this way, the user has the option of exploring the effects of incrementally changing one of the variable values. This is achieved as painlessly as possible through the use of a pre-processor. The user instructs the program to change the targeted variable from some particular value to another. The user also specifies the increments of the change. For example, changing the wall reflectance from 20% to 40% to 60% to 80%. The pre-processor then sets up the required files for a batch run.

If after using the pre-processor to set up incremental comparisons the user wants to review the input data, the target variable display will contain the letters "INC" instead of a specific value. Other variables that are effected by this variable will also be labeled "INC". An example is the case when the window width is incrementally increased. This means that the distance from the edge of the window to the corner of the wall will decrease. This feature lets the user know what other variables are dependent on his target element, as well as keeping him from inadvertently changing one of these other variables and producing garbage results.

Because the calculation time increases dramatically as the number of desired points for illumination values increase, the user is also given the option by the pre-processor to chose how many points he want to have calculated. This way he can cut down the run time tremendously so that a complete batch run will only take a few minutes.
After the illumination levels have been calculated, the user has the option to view the results in the same manner as the original MICROLITE 1.0 if desired. These choices include a plan view with numerical values at the analysis point locations, sectional graphs with the linear trends for a series of points, and axonometric displays with the lighting levels shown in a chain-mesh like manner. If the user chooses the pre-processor to set-up the incremental changes in a particular variable, he can also invoke the post-processor to set-up the comparisons of the relative effects of these changes. These variations in lighting levels for any particular point, as well as series of points, are sent to LOTUS 1-2-3 to be graphed using the bar and line graph options.

To be able to implement these types of features, a programmer naturally has to plan for them when he is originally coding the program. However, if he constructs the code to have an emphasis on modularity and self-contained sub-units, then the ability to add new, unforseen options at a later date is made much easier. In the case of MICROLITE 1-2-3, each major component of the program is separated and well labeled. These segments include: the data acquisition portion; the pre-processor containing the run parameter set-up; the various stages of the actual calculation procedures; and the results display and interpretation portion.

This modularity is also apparent in MICROLITE 1-2-3's heavy use of internal flags. Many of the display and calculation options selected by the user during the pre-process phase are represented within the program as boolean-like flags. This permits the program to make quick tests of the status of the flags in order to direct the internal flow of control. The pre-processor sets the flags
based on what decisions the user makes, and the program is able to proceed with the subsequent displays and calculations smoothly.

The display of the "INC" label is one example. For each variable that will be incremented during the analysis run, the processor sets the appropriate flags in the ABC data structure ahead of time. When displaying each variable value to the user, the program simply tests the status of the variable's flag. It doesn't need to go out and search for the user's decisions and the relationships among the variables, it simply checks the flag instead. This way, the conditions that lead to the triggering of flags don't have to be known to sections that don't require the information. It's not important why some choice was made, just that it was made.

This feature makes future updates and modifications of the processor easier. If conditions change for the selection of variables, or the relationships between them are altered, the main body of the code won't have to be rewritten. The only new coding would involve the processor and how the flags are set.

Another coding feature of MICROLITE 1-2-3 is the major use of structures with "semi-english" tags. Instead of having criptic arrays that look like "rm[1], rm[2], rm[3], etc.", this program uses "room.height.cur" to express the current value for the room height. Labels are also defined numeric values so that the code reads easier. For instance, "if (wall == NORTH) ..." is more understandable than "if (wall == 3) ...". These practices will make future updates easier because the programmer will be able to coordinate the flow of control with the daylighting calculation algorithms better.
A possible drawback to this method is that the code can become rather lengthy. The number of characters per line may become quite large because of the long variable names. This in turn will increase the size of the entire program. However, as compilers improve in their ability to handle larger statements and files, this should become less of a problem. It is still a tremendous benefit for the programmer to be able to turn to the middle of a stack of code, and immediately understand what labels stand for what information and where in the analysis process he is looking.

The features of MICROLITE 1-2-3 that are an improvement in the user-program interface for educational purposes include: flexibility of information entry; targeting of user defined variables for more specific study; ease of iterative comparisons through the use of automated batch runs; choice of analysis detail provided by the option of picking the number of analysis points; and flexible study of the results through various graphs and iterative run comparisons.

A troublesome characteristic of the program involves the issue of trading one set of constraints for another. It is a problem related to the general trend toward the use of the AutoCAD software for graphic editing. When a user enters a room configuration on the AutoCAD package, he has to use specific attribute commands and precise syntax in order to properly store the information needed in the analysis. This sequence is in effect an aberration in the process of pursuing an idea. It is really a substitution of one limitation on the ability to act on a thought for another.

Just because it is possible to transfer information in this manner doesn't mean that it is always appropriate to do so. For analysis programs used more in
production, it is most likely a beneficial feature. In programs where the issue is strictly facilitating a spontaneous approach to learning, it is an inhibiting feature that probably shouldn't be encouraged.

A situation where tagging is probably acceptable is when a program is used in an intermediate production/education role. An example of this type of arrangement is when analysis programs are used in a design studio. If AutoCAD is being used as a design medium, the option of directly sending the design to a program for energy analysis or daylighting analysis is quite beneficial. As the design progresses, the reasons for the placement of the elements become more complex, so the simple flipping of alternatives is not such a trivial action anymore. The more information each entity can carry, the better.

Another reason why the AutoCAD link may not be appropriate is because the MICROLITE 1–2–3 program can only handle simple spatial configurations. This is basically because the algorithm structure in the original MICROLITE 1.0 is set up like this. The simplicity constriction is not necessarily bad because the effects of variable changes become more visible to the student. However, because AutoCAD has the capability of permitting the creation of quite complex spaces, the intended use and the actual use do not match efficiently. The software is not utilized to its full potential. The student may have become accustomed to drawing intricate configurations on the AutoCAD editor, but now has to restrain himself to simple rectangular boxes.

All told however, MICROLITE 1–2–3 is an improvement in the attempt to fit a program into the "educational" architectural environment. It provides the ability to test hypotheses about the incremental effects of variable changes
relatively easily. The user doesn't have to do the manual file set-up and run for each specific increment. This process is automated so that the student doesn't have to concentrate as much on computer syntax, and the amount of time it takes to set up and perform the run is much less. He has more freedom to explore an idea without having to consciously "think" as much about the actual procedures involved in performing the test.

The black box characteristic has also been lessened. The student is able to expose some of the underlying relationships effecting lighting levels through the use of iterative runs and comparison graphs. The user is not restricted to a pre-defined format to follow when testing different variable values. He may experiment with them in any order or for whatever reason he wishes.

It is through this procedural transparency that the underlying algorithms surface. Even if they are buried deep within the code, the user can begin to discover the trends and relationships of the variables through the iterative comparisons. It is through these sequential runs that the program fits better into the architectural design process.

When these results are presented, what becomes important are not the numerical values in particular, but the trends and relationships that develop between them. Whereas production programs are quite concerned with accuracy in the final numerical result, in education programs there is a little more leeway. The educational program is not going to be the tool that provides the numbers that major financial or production decisions are going to be based on. Instead, it is a tool that will hopefully develop intuitive, maybe even hazy understandings in the student. Extremely precise numbers are not required.
Many times the particular algorithm has a certain bias in the method of it's analysis. Even when all spatial, material, and environmental data are the same, different analysis techniques will produce different results. (25) But even in these cases, the trends stay the same. If the user can internalize an understanding of these trends, then some real learning is taking place.

What then becomes a factor is how this relational information is presented to the student. It needs to be displayed in clear graphics, and ones which show if any trends are even developing at all. The program shouldn't force the user into making certain conclusions, but should display the information in such a manner that what conclusions can be made, will be made.

Some immediate improvements to MICROLITE 1–2–3 that can be overlayed relatively easily, involve developing the processors further. One feature for the post-processor would be to perform a statistical analysis of the different incremental comparisons. The processor could find the changes that had the most effect, or least effect, on the lighting levels. It would be able to point out these findings if the user wishes. If the manner of testing different design features was hap-hazard, this could be a valuable aid.

These results could also be passed to the pre-processor so that the user in follow-up sessions could ask for the earlier statistical results to be displayed. He could then decide on what general direction would be the best to pursue. This way time spent on exploring inaccurate concepts could be minimized.

Another feature would be to provide a "reference manual" in the post-processor. This would contain information about lighting levels recommended for various types of spaces. For instance, the student would be able to ask if a particular
value is acceptable for people sitting at reading tables in a library. Information about other lighting factors could also be available. One question might be if there is a potential glare problem when a certain wall reflectance is used next to a window. This manual should stimulate the user's creativity by simply responding to the inquiries. It should not try to take away from the student's perogitives by directing the questions.

These additions are feasible, and would greatly increase MICROLITE 1–2–3's effectiveness as a learning aid. The basic format and character of the original program are still evident. However, in the process of discussing the concepts that form the basis of this paper, a new framework for the Microlite-type analysis begins to emerge. This next step in the development process responds to the pros and cons of the programs presented thus far, and provides a new foundation for future criticism.
PROPOSED PROGRAMMING FORMAT

"It is important when confronted with a number of facts, to sort out which are incidental and which are important."

Sherlock Holmes to Dr. Watson
EDUCATIONAL CONTEXT

Educational programs are a slightly different breed than production/education programs. Their purpose should be more direct and perceivable. They are there simply to help the user learn basic concepts about a design issue in the most effective manner possible. They should initially communicate to the user through their format that they are focusing on a particular aspect of design, and how the student can get a better understanding of it.

For pure education programs, the characteristics that I feel they should have are: nearly instantaneous processing of the data; encourage the running of parametric comparisons; have a simple but thorough generic analysis configuration; provide user with the option to specify the level of detail of analysis; utilize various forms of mechanical and procedural methods of entering information into the program; implement graphics that get to the point by conveying the relationships, information, and data accurately and precisely; supply various methods of representing the analysis results; provide automated incremental testing; facilitate statistical interpretation of the incremental runs to highlight trends; and implement processors to provide suggestions or to act as a consultant.

These are the categories that educational programs should address. Exactly how they are carried out will depend on the specific problem being explored. A package looking at structural analysis will be different from one experimenting with daylighting levels. Ideally all of the features will be developed evenly so that all aspects of the program work together.
The instantaneous processing of data is important because the student should be able to get a result of his action while the idea is still fresh, or active in his mind. The bonding of the action with the result is more effective. In addition, the speed of processing would facilitate the reflection-in-action process in that the user would be able to immediately form a new understanding of the situation. The hypotheses, results, and conclusions are all in working memory at the same time, so that it is easier to form a new, modified hypothesis.

The generic situation is important because it allows the student to concentrate on the major elements that effect the results. As the student's knowledge grows, the situations can become more complex. At least initially though, the configurations should be simple. This permits a sort of "everyman" relationship in that the understanding of the generic case builds foundations for all later learning, and can then be applied to more complex situations through the development of these generic heuristics.

Providing the user with the option of specifying the level of detail of the analysis is an attempt to allow the program to be more adaptive to the student's interests. This is in recognition of the fact that analyses usually take a fair amount of time to run. This feature allows the student to calculate the values for whatever number of points that fit best into his time constraints. Some analysis cases run faster, some slower, so the number of choices should vary according to what is most appropriate for that particular program.

Utilizing various forms of mechanical and procedural methods to enter information into the program can be accomplished by using mice in addition to the keyboard. Someday even voice recognition could be an option. The
keyboard actions should take place with as minimal keystrokes as possible, and the screen locations for the mouse sequences should be laid out to permit a smooth visual flow from one selection to another. The procedural format can be more efficient when commands can be expressed in a number of ways. It could be noun-verb or vice versa. It could be part keyboard, part mouse, or maybe even part voice.

Graphics is a key aspect in that all of the information is somehow transmitted to the user through visual means. The structure of how the data and results are presented to the user is critical to the understanding of the situation. The screen design should be clear, and the important bits of information should be quite accessible visually. There should not be a lot of extraneous stimuli or visual clutter to detract from the pertinent information. The graphics for spatial or tabular representations should be clean, and contain only the necessary "tricks" to convey the data better. (26)

To facilitate the user's particular interest in looking at a specific element and testing it, there should be various methods to represent the data. Whether this should be a series of graphs, three dimensional representations or sectional views depends on the information to be transmitted and what context it is in. The user should have the option to choose, because people approach problems from any number of vantage points and no one particular method of inquiry is necessarily better than another. The program should be flexible enough to accommodate this.

Incremental testing of variables within a particular configuration is a valuable way to uncover the underlying relationships in that setting. A program should support and encourage this because it is an integral part of the learning process
of architectural design. The mechanics of the testing should be fairly well automated and easy to implement. Incremental testing reveals what variables participate more effectively than others in the analysis outcome.

To help the student perceive these "participatory" trends, a program feature that statistically analyzes the results of a series of runs would be beneficial. When a student experiments with a number of iterative runs in a non-directed order, and if the method of inquiry does not specifically address a particular idea, then some inherent relationships that do exist might not readily surface. But a processor that presents the results in a more coherent format would encourage the user to perceive these underlying relationships.

Finally, a processor that acts as a consultant for the student is a valuable addition. Questions about code requirements or recommended values pertaining to different contextual settings could enable the student to relate the new conceptual understandings to real life design considerations. One possibility is a knowledge base environment that fetches the pertinent data directly from the analysis configuration, and only then questions the user for new information needed to derive the answer or recommendation.
THE INTERLITE PROGRAM

An example of what a purely educational program could be like is Interlite. It is a progressive development of the MICROLITE family of daylighting analyses. It would be an interactive tool hopefully as natural and unintimidating to use as is a television set. It would provide the architecture student with the opportunity to play around with an interesting package and learn something at the same time.

One of the major features of INTERLITE would be the design of the screen interface. All of the major information pieces involved with the analysis are displayed at the same time. This would be accomplished using windows, and the ability of some extremely fast CPU’s.

The left-hand portion of the screen would display an axonometric view of a simple spatial configuration. In the upper right-hand corner would be a line graph of the relative lighting levels for whatever section the user has currently selected. Right below that, in the right mid-section, would be a floor plan plot of the numeric lighting values. The lower right portion of the screen would contain the labels and current numeric values of the external site conditions. The area for traditional keyboard questions and answers would be located along the bottom left-hand edge.

The room configuration being analyzed would be displayed as an axon. Within the axon, the variables that comprise the space would be shown. The alphabetic labels with their current numeric values would be displayed in half tone. The student would be able to alter the values of these variables through the
keyboard. He could type the variable name and the new value, or type a positive/negative sign and the amount of change from the current value.

An additional way to change the variable would be to use a three-button mouse. The variable would be activated by positioning the screen cross-hairs of the mouse over the label and clicking one of the buttons. Once the label is "turned on", the user can lower the value by pressing one button, and raise it by pressing another. This procedure could act like the scrolling feature offered on many photocopiers to select a magnification/reduction level.

As the wall length and room height variables change, the shape of the space in the axon should be modified. The same would happen for the size and proportions of the windows. This feature would allow the student to observe the changing spatial configuration visually.

At the same time as the axonometric is being updated, the graphs along the right-hand side of the screen should also. Ideally this update would be instantaneous. This is where ultra-quick processors or multi-processors would be required. (28) The program would calculate all of the numeric values for the array of analysis points, and then update the section graph and the numeric floor plan graph. If the user desires, a chain-link type mesh could also be placed over the axon to show the lighting levels, similar to the feature of the original MICROLITE 1.0.

The speed of turn-around in the results would be a tremendous benefit to the student. He could see what the lighting values are for the altered spatial configuration while the thought that drove him to make that change was still clear in his mind. Making changes is thus encouraged by the program because
it is so easy. This facilitates the characteristic hypothesis and test element of architectural design and knowledge acquisition.

So far the program has been described to handle situations in "real time". However, comparisons of different related configurations should be possible. Much in the same way that MICROLITE 1–2–3 uses the pre–processor to increment the variables, INTERLITE could do the same.

When the user activates one of the variables using the word "increment" ("inc" for short), he would have access to the incremental run feature of the program. The program would prompt the user for the initial starting and ending values of the variable. Default values could also be accepted. These values could be entered through the use of the keyboard or the mouse with scrolling. The program would then calculate the lighting levels for each unitary increment of the variable in–between the starting and ending points.

The change of the lighting level in any of the analysis points could then be shown in the upper right–hand, sectional graph by triggering that specific point in the floor–plan graph. The user could do this by typing in the command, "graph X", where X is the "coordinate" number of the point displayed in the floor–plan graph. For more detailed comparisons, the student could select an overlay window which would fill nearly the whole screen. This would make it easier to compare the incremental results for a series of points.

The visual representation of the analysis space and of the analysis results are key components to the INTERLITE proposal. The information in these displays should be accessed directly through the use of mice to further increase this visual connection. This is important because architects tend to think visually. It
is advantageous to get all the information on the screen so that he can see it more readily to work with it. This in turn will help him retain the conclusions better. The spaces analyzed and the results obtained would then be easier to visualize and remember than would a series of numbers on a paper printout.

The whole process of using the program should be relatively simple and up-front. When the user thinks of a test to run, he should be able to "look" at the design element and easily run the test. The program would then be used more often because it is non-complicated and simple to implement. It is fun to test different ideas if the testing process is painless. Architects are generally quite inquisitive, so if the program encourages these comparisons it will probably be used a lot. It will atleast while the knowledge gained from it is still new to the user.

While this is only a proposal of what could be, it is a goal to shoot for when designing this kind of software. The major limitations to it's current feasibility is the speed of the analysis process itself. Microcomputers that are currently in production are not able to process information that fast. Some of the more sophisticated computers can, and certainly the Cray's are able to. But how realistic is it to expect someone to devote one of these supercomputers to a "trivial" program such as this? Besides, part of the concept of this type of software package is to be as unintimidating as possible. In this aspect microcomputers have a very distinct advantage over the large monsters anyway.
"Computers that 'teach' and systems that render 'expert' business decisions could eventually produce a generation of students and managers who have no faith in their own intuition and expertise."

H. & S. Dreyfus
CONCLUSIONS

The issues that have been discussed in this paper are both a synopsis of some of the existing programming efforts and concepts being implemented at this particular moment, as well as a recipe of ideas to stimulate further developments. The factors that will effect future programming improvements are many, and as study in areas such as cognitive psychology, artificial intelligence, and hardware technology continue, they will have a significant effect on the software development.

As programs develop, the inquisitiveness and creativity of designers is a trait that must be supported. The use of the computer should not detract from this ability. It should become a medium to stimulate new ideas and understandings. The processors of MICROLITE 1-2-3 are an effort in this direction. They enable the user to explore concepts personal to himself in whatever iterative fashion desired. They don't state what the "correct" conclusions are. Instead, they allow the user to think about the relationships and make his own decisions.

One of the emerging facets of the computer – architecture interface is the application of expert system technology to architectural problems. These efforts have ranged from floor plan design to code checking of human proposals. Expert systems also have the potential to act as tutors for design students.

The potential overlay of the knowledge base routines to the MICROLITE 1-2-3 program is one example of this expert system contribution. However, the use of expert systems need to be integrated appropriately within the design process. They shouldn't control the user's approach to a problem. Instead, they should
act as a resource for the designer to enable him to explore an idea in greater
detail, or from a different vantage point.

The authoritative aspect of many expert systems need to be avoided in this
type of environment. The systems are based on someone else's heuristics and
understandings, and themselves contain the biases and peculiarities of the people
that construct them. Authoritative systems may be appropriate in other
situations, but in this particular educational setting I don't feel they are. The
opportunity to stimulate creative thinking, not remove decision making from the
user, should be the priority here.

Educational programs must be visually and procedurally attractive to the user.
When a person is interested in the program analysis and actively participating
in the questioning process, the potential to acquire new knowledge is improved.
The purpose of the programs should not be to entertain, but a moderate degree
of pleasure should be achieved by using them.

These programs should take a lesson from video games. Video games have to
be exciting and appealing in order to be played often. They also have to
present additional challenges as the user becomes more proficient. Sophisticated
graphics and fast response times are characteristics of successful video games.

Educational programs should be the same way. They should be "fun" to use by
implementing interesting graphics and processing the data quickly. They should
also not require a lot of procedural actions and impose restrictions. If they are
a "drag" to use, they won't be used. The Wordstar tutorial is notoriously slow
and boring to run. Consequently, students soon quit using the tutorial series
and jump right into using the editor to explore and learn the features.
Industry will not be the driving force in this type of software programming because there is not very much money in it. There are many more architects as potential buyers than there are schools. If an architect feels that some particular package will help him increase his profits through more effective marketing, quicker design time, or improved design quality, then the architect will buy the software. Computer programmers know this, so they are targeting the practicing architect as clients. The bottom line in business is money, profit.

This leaves it up to the universities to program this type of educational software. (29) It is one based more on a conceptual and research oriented approach. There simply isn’t enough money out there for the profit driven programmers to survive writing educational software. However, an interesting symbiotic relationship is developing between industry and the universities. While the basic business principle of increasing sales lies at the heart of the matter, the effects will be of furthering the efforts to program software for educational purposes.

What is happening is that some vendors, such as Autodesk of California, creators of the AutoCAD graphic editor, are making some of the internal file structures of their programs public. What this is doing is providing links for other programmers to write their custom software to interface with these existing packages. The DXF file is one such “hook”. While third-party programmers in industry are writing programs utilizing these links, programmers in architectural schools are also reaching for this bait.

MICROLITE 1-2-3 is one example of this relationship. The concepts of improving integration and making the data transmission sequences more flexible is furthered by the linking with AutoCAD (via CRL) and LOTUS 1-2-3.
AutoCAD is also becoming known to more future architects and architectural computer programmers at the same time. AutoCAD is thus becoming a more integral piece in the network of software development.

The future improvements in other areas of computer and computer-related technologies could also have potential benefits for this educational context. Multiprocessors could greatly increase the computational speed and responsiveness of the programs, and computer generated holographic images could enable the user to actually perceive the theoretical space in three dimensions. The improved speed and computational power would mean that the more sophisticated graphic shading algorithms could become more feasible to implement.

Whether educational daylighting analysis programs will be given immediate access to these progressive technologies may be another story. As the economic and political climates shift university research more toward applied "real world" concerns and away from pure academic exercises, educational daylighting software may be placed on a low priority level. There is a greater emphasis on projects that can return some sort of direct, tangible benefit to the institution. A major factor is the monetary funding by outside sources. Therefore, the access to these new technologies could be dominated by such industries as defense, nuclear, and medical. Unless there is another energy "crisis", or utilities place a higher priority on preventative energy use, educational daylighting software will probably not become a dominant issue.

The effort to improve software for use by architectural students will continue however. The potential for improvement is too great not to be addressed by programmers. The funding may not be spectacular, but as M.I.T.'s Project Athena (30) demonstrates, it is available. The process of developing this type
of software is in itself a tremendous education, and as more people experience this procedure, the benefits will propogate throughout the architecture industry.
APPENDIX - EXCERPTS OF MICROLITE 1-2-3 CODE

#define WIDTH 0
#define DEPTH 1
#define HEIGHT 2
#define THICK 3
#define RCEILREF 4
#define RWALLREF 5
#define RFLRREF 6
#define ANALHIT 7
#define ROOMCOMPO 8
#define HIGH 0
#define LOW I
#define CUR 2
#define PREP 3
typedef double real;
double room[ROOMCOMPO][4];
char *mesg[] = { /* RM+0 */ "Room data messages",
 "Room data",
 "Target variable value set by pre-processor",
 "Room width (north/south walls): ",
 "Room depth (east/west walls): ",
 /* RM+5 */ "Room height: ",
 "Wall thickness: ",
 "What is the room width (n/s walls)? %%s",
 "What is the room depth (e/w walls)? %%s",
 "What is the room height? %%s",
 /* RM+10 */ "What is the wall thickness? %%s",
 "What is the ceiling reflectance (%%)? %%s",
 "What is the wall reflectance (%%)? %%s",
 "What is the floor reflectance (%%)? %%s",
 "What is the analysis height? %%s",
 /* WM+0 */ "Window messages",
 *****************************************
 ,
 ,
 /* Window data ",
 ,
 ,
 /* WM+5 */ "South wall",
 "West wall",
 "North wall",
 "East wall",
 /* WM+10 */ "Want to enter a window on this wall? %%s",
 "Want to enter another window? %%s",
 ...
};
/* entry and display sequences of room data */

room_measurements() {
    double tcur, tlow, thigh;
    int i;

    initscr();
    echo();
    while(TRUE) {
        for (i=0; i<5; i++) {
            move(i,0);
            puts(msg[RM+1+i]);
        }
        refresh();
        for (i=0; i<ROOMCOMPO; i++) {
            move(8+i,0);
            if (room[i][PREP]) {
                /* variable targeted for */
                /* increments by pre-processor */
                puts(msg[RM+6+i]);
                move(8+i,ANSLOC);
                puts(INCREM);
                refresh();
            } else {
                if (i==RCEILREF || i==RWALLREF || i==RFLRREF) {
                    tcur = room[i][CUR] * 100;
                    tlow = room[i][LOW] * 100;
                    thigh = room[i][HIGH] * 100;
                    tcur = getreal(msg[RM+6+i],
                                    tlow, thigh, tcur, 0);
                    room[i][CUR] = tcur * .01;
                } else
                    room[i][CUR] =
                        getreal(msg[RM+6+i],
                                room[i][LOW],
                                room[i][HIGH],
                                room[i][CUR], 1);
            }
        }
        if (infocorrect()) {
            endwin();
            break;
        } else {
            clear();
            continue;
        }
    }
}
window_data() {

double tcur, tlow, thigh;
int l, wall, win, wmt;

for (wall = 0; wall < 4; wall++) {
    win = 0;
    while(TRUE) {
        initscr();
echo();
        for (i = 0; i < 5; i++) {
            move(i, 0);
            puts(mesg[WM + 1 + i]);
        }
        puts(mesg[WM + 6 + wall]);
        refresh();
        proceed = FALSE;
        newwin = FALSE;
        if (win == windo[wall][windows] &&
            win < MAXWIN) {
            if (windo[wall][windows] == 0)
                wmt = WM + 10;
            else
                wmt = WM + 11;
            move(10, 0);
            if (getintgr(mesg[wmt], (long) 0,
                            (long) 1, (long) 'n')) {
                proceed = TRUE;
                newwin = TRUE;
                windo[wall][windows]++;
            }
            move(10, 0);
deleteln();
        }
        else if (win < windo[wall][windows])
            proceed = TRUE;
        }
    }
}
if (proceed) {
    while(TRUE) {
        move(8,0);
        printf("Window %ld of %ld", win, window[wall][windows]);
        move(10,0);
        clrtoeol();
        move(13,0);
        puts(msg[WM+14]);
        move(12,0);
        puts(msg[WM+13]);
        refresh();
        move(11,0);
        if(window[wall][win][type][PREP]) {
            puts(msg[WM+12]);
            move(11,ANSLOC);
            puts(INCREM);
        } else {
            window[wall][win][type][CUR] =
            getreal(msg[WM+12],
            window[wall][win][type][LOW],
            window[wall][win][type][HIGH],
            window[wall][win][type][CUR],
            0);
        }
        for (i=0; i<2; i++) {
        }
/* transmission & reflectance */
move(14+i,0);
if (windo[wall][win][TRANS+i][PREP]) {
    puts(mesg[WM+15+i]);
    move(14+i,ANSLOC);
    puts(INCREM);
    refresh();
}
else {
tcur=windo[wall][win][TRANS+i][CUR]*100;
tlow=windo[wall][win][TRANS+i][LOW]*100;
thigh=windo[wall][win][TRANS+i][HIGH]*100;
tcur=getreal(mesg[WM+15+i],tlow,
thigh,tcur,0);
windo[wall][win][TRANS+i][CUR] =
tcur *.01;
}
REFERENCES


3. Ibid.

4. Ibid.

5. Ibid., p. 12.

6. Ibid., p. 64.

7. Ibid., p. 7.

8. Ibid., p. vii.


10. Ibid., p. 24.


15. Ibid., p. 159.


23. *Ibid.*.


27. Discussions with Harvey Bryan.


30. M.I.T. Project Athena is an attempt to integrate the computer into all phases of higher education. Funding for research and programming projects is donated by non-academic sources.
BIBLIOGRAPHY


"Can you repeat everything from the beginning?"

Shubhada Bhave