A Visually Enhanced Programming Environment

by
Ming Chen

Bachelor of Fine Arts, California Institute of the Arts
Valencia, California, 1983

Master of Fine Arts, California Institute of the Arts
Valencia, California, 1986

Submitted to the Media Arts and Sciences Section
School of Architecture and Planning
in partial fulfillment for the requirement of the degree of
Master of Science in Visual Studies
at the Massachusetts Institute of Technology
September 1990

Author:

Media Arts and Sciences Section
August 10, 1990

Certified by:

Muriel Cooper
Professor of Visual Studies
Thesis Supervisor

Accepted by:

Stephen Benton
Department Committee for Graduate Students

A Visually Enhanced Programmming Environment (Program Design)

by
Ming Chen

Submitted to the Media Arts and Sciences Section
School of Architecture and Planning
on August 10, 1990
in partial fulfillment of the requirements for the degree of
Master of Science in Visual Studies.

Abstract

Existing programming environments suffer from the lack of perceptual navigation capabilities. This thesis addresses that problem by proposing a programming environment in which the organization, access, and presentation of the program stresses visual (boundary, position, shape, color, and text) instead of structural hierarchies (directories and files). The anticipated result would be programs that are easier to design, understand, debug, and modify.

References are made to the visual design principles and techniques used in maps as a guide to the research, since navigation, whether through information or geographically, ought to share common principles.

Thesis supervisor: Muriel Cooper, Professor of Visual Studies

This work was supported in part by Hewlett-Packard and DARPA.

Acknowledgements

I am most grateful to Professor Muriel Cooper for giving me the unique opportunity to enlarge my knowledge at the Media Laboratory, and for her perceptive guidance in my work.

I am also so indebted to Louis Danziger for my design education and paternal guidance; and to Professor Donald A. Schon for his insprational perspective on design.

At the Visible Language Workshop, I am blessed with friends whose help have made the critical difference in my work. For her ever dependable assistance, friendship, and care, I owe immesurable gratitude to Laura Robin; and sincere thanks to Suguru Ishizaki for making my entry into the laboratory a smooth transition, and the great conversations on design and technology.

For reviewing this thesis, ideas and advice, Richard Rubenstein, Henry Lieberman, and Ronald MacNeil.

For administrative assistance and friendship, Marie Crawley and Anne Russell. For her understanding, and sympathetic ear, Linda Peterson.

For technical assistance, advice, and lasting friendship, Sylvain Morgain, Bob Sabiston, David Small, and David Young.

Most of all, I thank my Parents, for all their sacrifices.

August 1990
Cambridge, Massachusetts

Contents

1.0 Introduction

Of all our senses it is vision that most informs the mind. The instruments of science also favour vision, but they extend it into new domains of scale, intensity, and color. . .

Powers of Ten

1.1 Background

At the Visible Language Workshop, improving the comprehension of information through the fusion of visual design principles and techniques, with technological innovations, is the unceasing endeavor. Computer technology introduces new issues for the graphic designer to grapple with. The most important of which, is dynamics (including interactivity, and personalization), demands different considerations than the static medium of print. Dynamic information, that is, information which can change and move, has broad applications in such diverse domains as scientific and software visualization, signage and navigation, electronic publication, and "on-line" information services.

Instead of the interaction between the user and machine; it is the interaction between the user and information, that is, the communication process, that we are interested in. We seek ways to improve the effectiveness of visual communication by studying the inherent properties of information and human behavior, and to apply this knowledge to the various domains of information design stated above.

1.2 Problems With Existing Programming Environments.

As computer technology is applied to increasingly ambitious problems, its complexity grows ever faster. The perceived complexity of an information system is not just a simple function of the amount of information in it. Even complex systems can be made manageable, if the information is organized in a logical manner, if important relationships are clearly presented, and if relevant information is easily accessed.

While there has been much progress made in user interface techniques and environments, their usability* has been often overlooked. This may be because the tools are produced by computer scientists, who do not know enough about communication and visual design.

* Usability is defined as the convenience and practicability for use, including visual refinement.

Existing programming environments suffer from the lack of intuitive navigational capabilities for the user. Most programming environments organize a program into a static hierarchy, divided into directories, files, and functions. The functions within a file are sequentially arranged, making the split-buffer effect the only way to simultaneously review functions not placed close enough to be displayed on one buffer. Programmers have to commit to memory a 'mental picture' of the program as they navigates through it, visiting individual functions. Much could be learned about the relationships between the functions by the mere spatial juxtaposition of the functions, or groups of functions. In a traditional programming environment, the functions in a file are shown in their entirety: code and documentations all; without the capability of selective display of information. Another disadvantage is the lack of any visual sophistication such as color, translucency, or proportionately spaced text.

In summary, the main identifiable problems of traditional software environments are:

1. context/continuity,
2. locality of information,
3. levels of detail,
4. clutter, and
5. visual quality.

Problems 1 - 4 are similar because they share common inadequacies which are spatial in nature. Problem 5 has different concerns, mainly with the manipulation of the details of form. Compared with spatial manipulation, the manipulation of form - color, scale, shape, texture - are secondary, belonging to the sphere of the mastery of details.

The user-machine communication requires a way of thinking which most humans are not accustomed to. Images are virtual, and manipulation of the images is by proxy, through an "unintelligible" computer language. Not only do programs have to serve as the communication between human and machine; they also frequently have to be understood between humans as well (such as, reading someone else's code, or even one's own).

1.3 The Goal of The Thesis

This project explores the idea of using a visual hierarchy, utilizing the principles and techniques* of visual design, instead of a structural one of directories and files, to organize a program. A structural organization restricts navigation through a program along a strict protocol of access paths (e.g., in UNIX /file-system or partition/directory/s-bdirectory/file). Another way of describing the problem and the intended goal, is to think of having a whole program reside in a single file, something beginning programmers tend to do. This technique is ghastly only within the restraint of a visually impoverished display environment. With improved display technology, and selective use of visual design principles and techniques, the idea is actually quite attractive, make programming environments more effective in terms of human cognition, and how the information is organized, accessed, and presented. The anticipated result would be programs that are easier to design, understand, debug, and modify.

* Design principles are largely built upon the principles of the Gestalt psychologists because they share the same concerns of perception. The difference lies in that psychologist are interested as why humans experience such perceptions, whereas designers are mainly interested in how to exploit or avoid such phenomena in the work they produce: closure, continuance, organization (by proximity, similarity), 'common fate', and contrast, are among the more important ones.

1.4 The Approach

The visual design of programs should be approached from both the practical needs of programming, as well as from a more theoretical viewpoint of the design of dynamic information. There is no theory-based set of design principles that deals with the design of dynamic information. By using graphic design as a window, it is possible to identify the perceptual elements involved, and to discuss some principles of "good fit" between physical form and the content/context of the information to be visualized in a way that will supplement the process of developing good, effective visual representations.

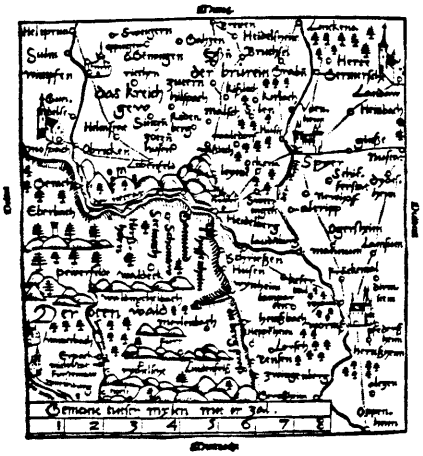
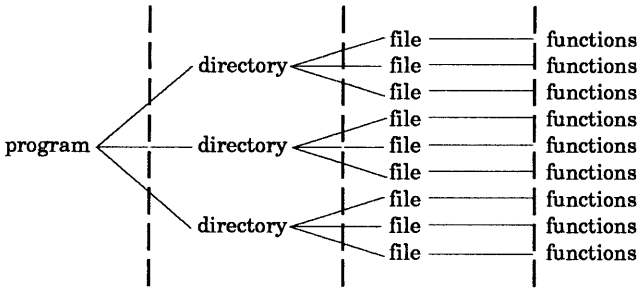
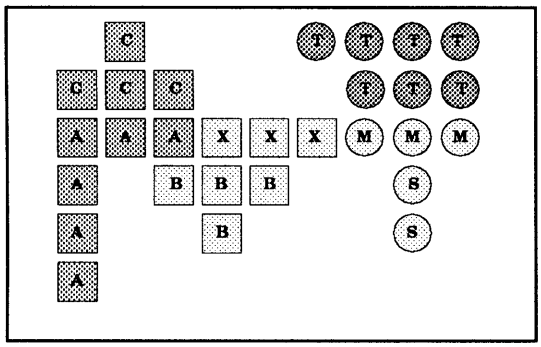


Figure 1.1 Ancient regional map of Heidelberg and environs (1528). *Graphis Diagram 1*, 1988.



(a)



(b)

Figure 1.2 (a) A structural hierarchy of information where each sub class is obscured by its superclass. Access to a sub-class is through a prescribed path. (b) In a visual hierarchy, objects are grouped by perceptual attributes, such as, boundary, color, shape, proximity, and text. Access to a group of objects is direct.

Do not compare a visual hierarchy to a structural hierarchy because their underlying structural organization in terms of the access of information is different. If a comparison *must* be made, the bounding box would be equivalent to the program, shape to directories, and color to files. Alternatively, color could be used as directories, and shape for files, in accordance to the findings stated in SECTION 5.3.1.

The map is an appropriate metaphor to study, because it was developed as a navigational aid. Navigation, whether abstract, as through information, or concrete, as geographically, ought to share certain common principles. The generalized ideas and visual techniques of map design could be adapted for the benefit of programming environments.

Finally, it is important to clarify that we are not attempting a *graphical programming* approach to the problem, although we share many of the same visual parameters. High-level programming languages, like LISP, do a reasonable job in making code understandable; nevertheless, there is still much room for improvement towards becoming more similar to natural language. Programming languages have evolved in their textual form, and it would be prudent to favor future developments in an evolutionary tradition, rather than enforcing a novel and untried method, which would create as many new problems as it would solve.

☞

2.0 Project Description

2.1 The Program Environment

Programming environments influence the style of programming. The conceptual foundation of our programming environment is the metaphoric 'flat' map, which stresses a visually and spatially oriented 'style,' allowing the easy lateral translocation from one part of a program to another, without losing contextual references to the overall program. This chapter discusses how the *map-program*, is implemented, and its use; and see how they enable us to overcome the problems associated with a traditional programming environment. A secondary task is to examine how this new approach to programming could be utilized to alleviate some of the frustrations associated with debugging.

2.2 The Structure Of The Programming Environment.

2.2.1 The Basic Unit: The Function

The basic building block for our map is the function. In an interpreted programming language, such as LISP, a program is seen as a collection of functions working together to perform a task. Upon evaluation, all functions are dumped into the big vat of the LISP environment.

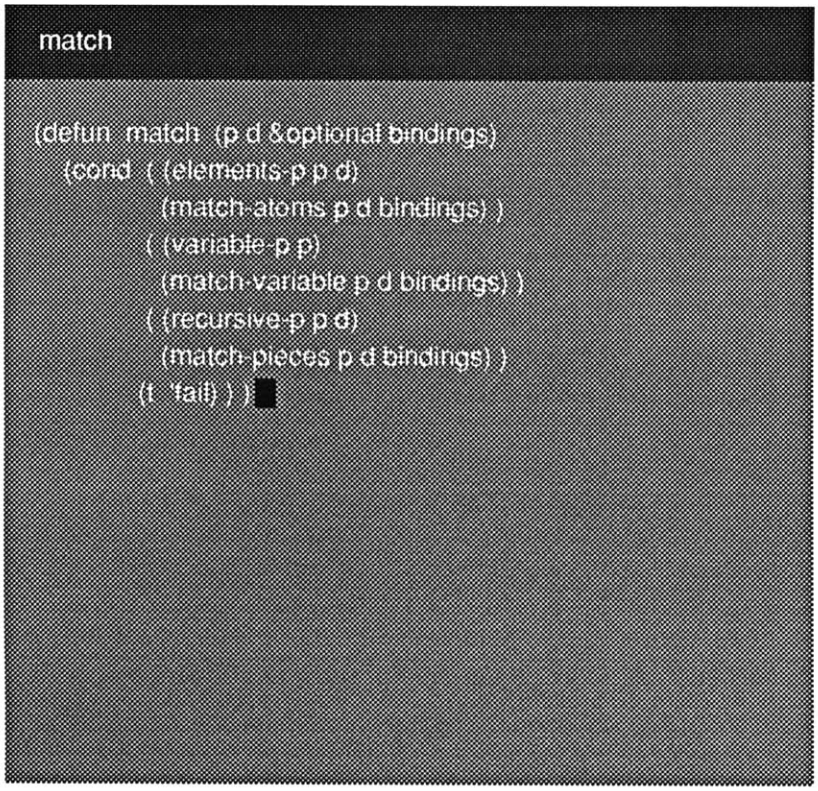
The visual manifestation of a function on the map is the *function-label* *. Its primary task is simply that of a place marker, that is, to ascertain its *spatial disposition* (SECTION 4.2.1) on the map, and in relation to the other functions. The label is the 'gateway' to the other components of the function.

The second major component of the function, is the coding window. Resizing (see FIGURE 2.1) the label, reveals the coding-window, initially empty, into which the user can enter the procedural instructions that make up the function. This technique could be

function-name

* A function-label is a function name on a rectangular (window) background. The rectangular background becomes the title bar of the coding window, as shall be discussed later.

Other components of a function can include documentation windows and flow-charts. The purpose of documentation windows is to separate out different kinds of documentational information, so that each could be selectively accessed (principle of *selective utility*, SECTION 4.2.4). This is preferable to having the documentations always being in the coding window cluttering up the space.



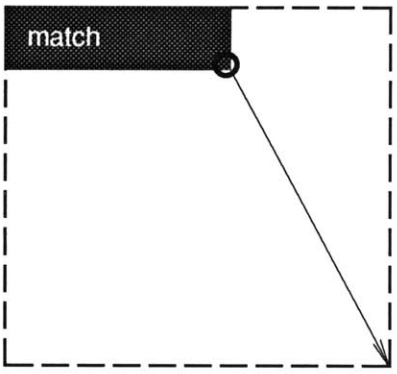


Figure 2.1. By resizing the function-label, the coding-window is revealed.

Figure 2.2. The components of a function. With direct-programming, code is entered directly into the coding-window.

called *direct-programming*. Thus, the code describing a function would be available at the same location of the function-labels (*locality of information*, SECTION 4.2.3). All editing to the code of the function would be performed directly in the coding window of the function concerned, and all future access to the code would be through its function-label. On command, the code within would be compiled, and dumped into the LISP environment. When coding for that function is complete, the coding-window could be closed to show just the label.

2.2.2 The Global Map - Context

A program can easily contain over a hundred functions. Even for a dynamic medium, to display all the functions within the map on a single screen could be difficult. In the map-program system, a two screen programming environment is used. On one screen is displayed the *global-map*, and the other, the *magnified map*. Actually, the map-program is only a concept that exists in our mind. The global map is a representation of the entire map-pro-

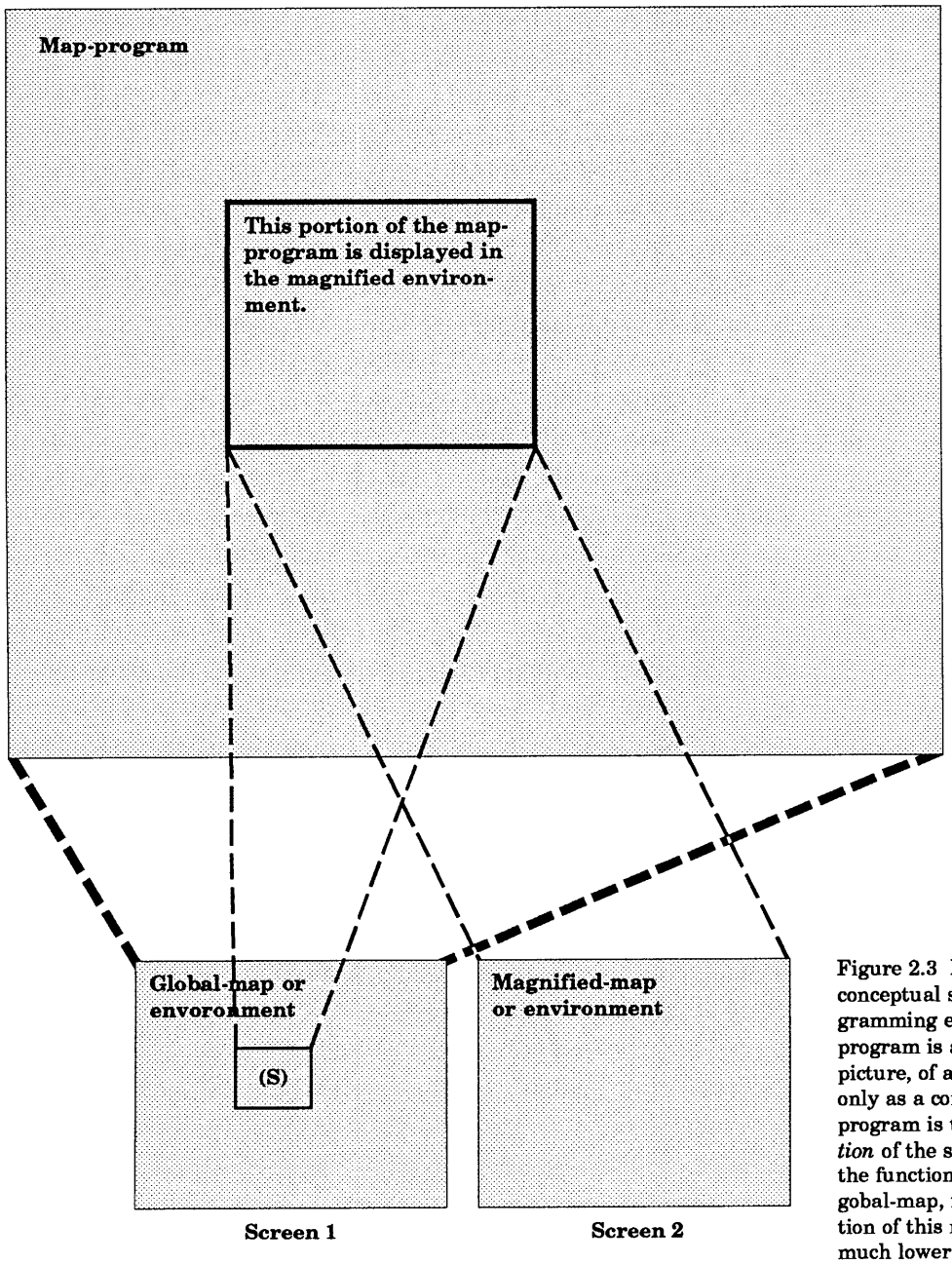


Figure 2.3 Diagram showing the conceptual structure of the map programming environment. The map-program is actually a virtual, mental picture, of an entire program. It exist only as a conceptual model. The map-program is the *conceptual representation* of the single file containing all the functions of the program. The gobal-map, is the *visual* representation of this map-program, but at a much lower information resolution. It's purpose is primarily to maintain the visual continuity, hence the context, of the program. The portion of the map-program demarcated by the scope-window, (S), on the global-map, is displayed in the magnified environment. It is in the magnified environment that all manipulations to a function are made.

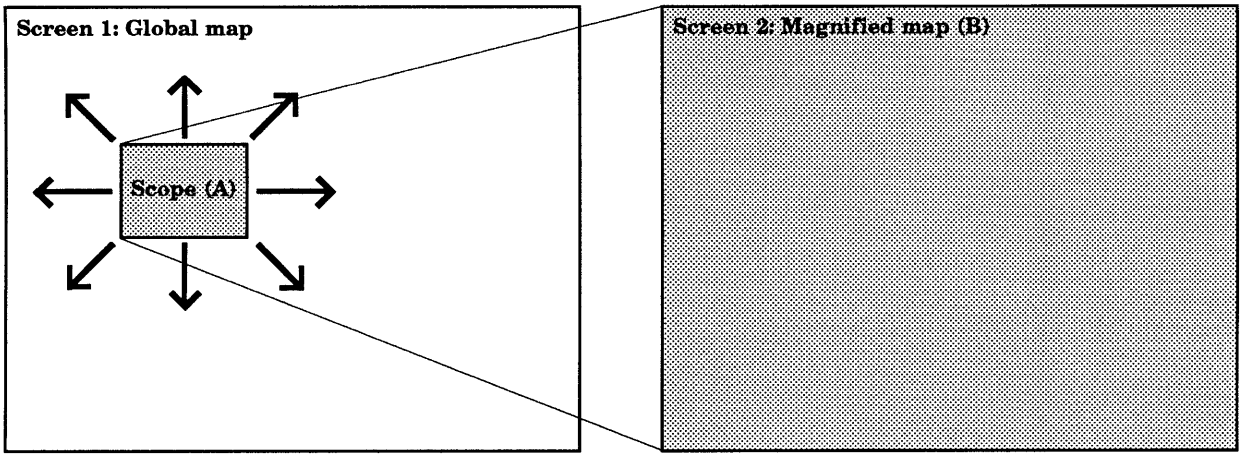


Figure 2.4 The magnified environment (B) displays in detail everything within the scope-window (A).

gram, displaying all the functions within a single screen. It is not a direct scaling down of the entire map-program.

The global-map captures, in a single glance, the inter-dependencies between the individual functions, or groups of functions - a bird's eye view of the overall program - without getting into details. Thus, it is a good *browser.* In establishing the context, the intention of the overall program becomes clear. To then skip from one function to another, is a relatively trivial matter. This makes the access of information simple and straight-forward.

2.2.3 The Scope Window And The Magnified Environment*

The scope-window is a transparent window on the global-map which behaves like a magnifier. The term "scope" means viewing, not scoping as in programming. The scope-window is free to move around the global map (see FIGURE 2.6a-e), and all that is within its scope is, theoretically speaking, displayed on the magnified environment in 'real-size,' at an 4:1 ratio. What is actually happening is that the section of the imaginary map-program demarcated by the boundary of the scope-window is 'visualized' on the magnified environment.

It is on the magnified environment that all programming activities and manipulations to individual functions are performed. Any changes to the magnified environment, such as, creating a new function, deleting an existing function, move, or change of color to the function labels, are subsequently reflected on the global-map. The global map is the organizational environment, and the magnified environment is the executive environment.

* Reference must be made here to the Spatial Data Management System (SDMS). Drawing upon established techniques in visual design and perceptual psychology, the Architecture Machine Group at MIT, in the 1970's, began experimenting with the creation of an information management system that exploits the user's sense of spatiality for the purposes of organizing and retrieving data: spatial data management system. *Spatial Data Management System*, Richard A. Bolt, Massachusetts Institute of Technology, 1979.

Cross-referencing the code of different functions that lie within a magnified environment is simply a matter of resizing the map-function labels (to reveal the code) of the respective functions. To compare two functions which are beyond the scope of a single scope-window, a second scope-window would be assigned. The screen housing the global-map is temporarily converted to display the second magnified environment. After comparison, the second magnified map is closed to again display the global map.

2.2.4 Grouping The Functions By Visual Hierarchies

To reiterate, the concept behind the organization of a program is the amalgamation of the 'single file program' and the map; no directories and files, so that all functions are visible simultaneously. In traditional programming environments, files and directories exist for organizational purposes. With a graphical display, the grouping of functions could be achieved by using non-structural techniques, most easily by color-coding (APPENDIX 5.3.1.2). Just as functions that are similar to each other are traditionally placed together in the same file; similar functions in the map environment would be color-coded in the same color. Shape would be another way to classify functions by, except that there is still no efficient method for computers to process shapes. Rather than a structural hierarchy, which restricts the access of information along prescribed routes, a visual hierarchy is set up in which the manner of accessing the information could be random. The visual hierarchy in order is: general position, color, and finally textual or semantic description.

The global map is also a visual mnemonic. With respect to human memory, recognition is easier than recall. The presence of the map acts as an associative prompt to assist recollection. Unless the user is very disciplined, or compulsive, it is very difficult to think of a name for a new function when first creating it. Later on, when the user wants to find it, he cannot recall it because he did not have the proper name for it in the first place. Color-coding provide a general 'key' whereby functions of similar utility could be searched. Thus, color-coding has 'content' significance by association.

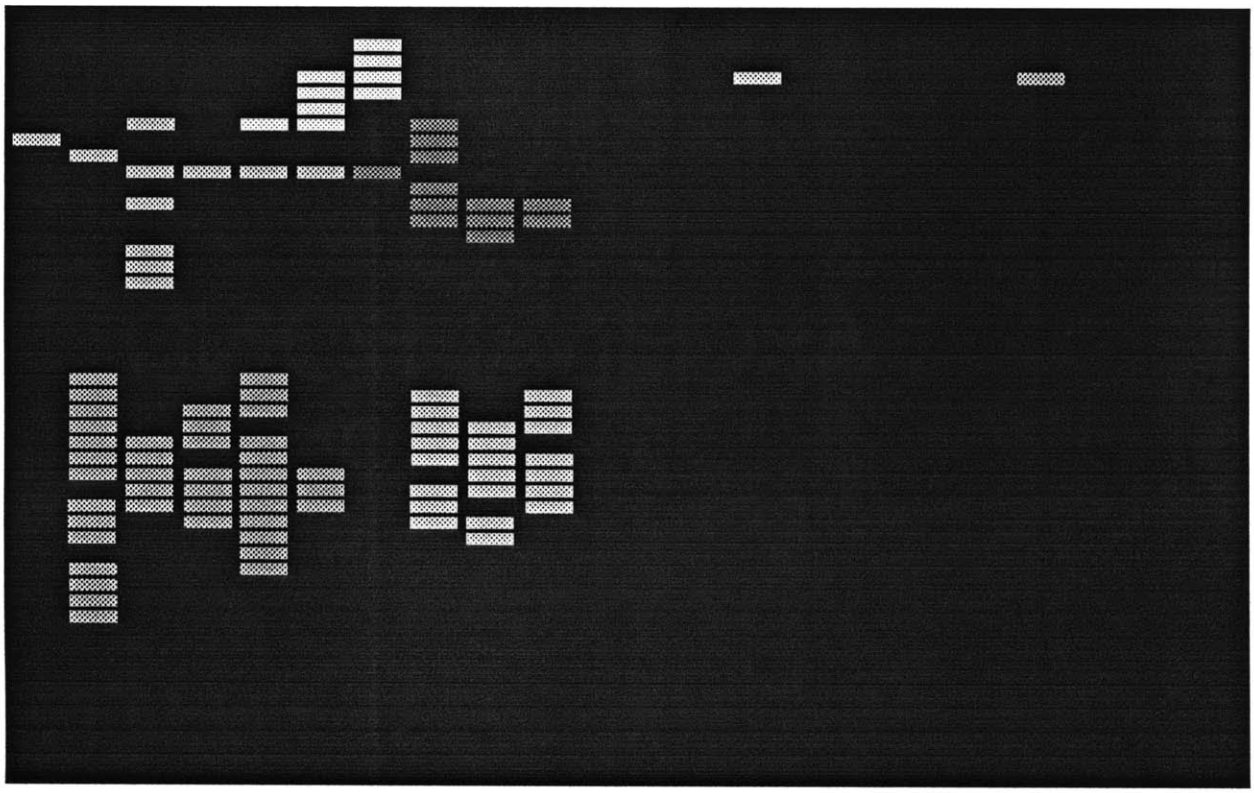


Figure 2.5a (above) A portion of the global-map showing the functions grouped by color instead of files.

Figure 2.5b (next page) A portion of the magnified environment showing clustering of function with similar utility within the same color-code.

2.2.4.2 Limitation Of Screen Space

The limitation on the number of functions that could be displayed on the global-map is determined by, 1) the maximum number of colors that could effectively indicate a meaningful difference between one group of function from the next, and 2) the physical space (screen real-estate) to display all the functions.

As a gross estimation of the size of program that the system can display and accomodate, the global-map, when completely filled, can display 2800 function-labels, leaving room for a title bar and menus. Suppose that, due to the spatial arrangement of the function-labels, only 10 per cent of the screen space is utilized, then 280 function-labels could be displayed. If each function typically contains an average of 15 lines of code, that would result in a 4200-line program, which could be considered a large program. Therefore, the map-program is a viable alternative to the traditional structurally hierarchical organization of programs.

stream-rest
stream-endp
stream-first
stream-cons
stream-member
stream-append
stream-transform
stream-concat
stream-remem

Global environment Magnified environment

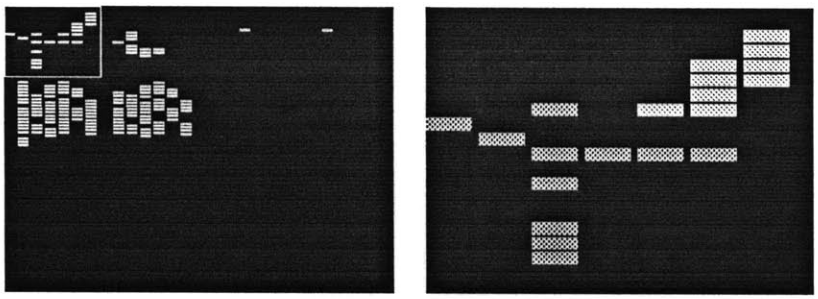


Figure 2.6a The global environment (left) holds all the functions in the program. Everything within the scope-window is displayed in the magnified environment (right).

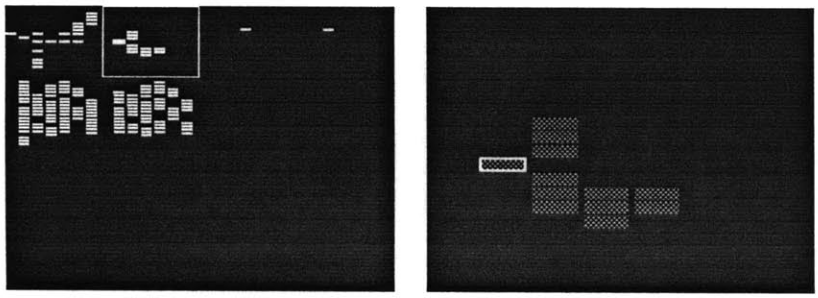


Figure 2.6b The scope-window is free to move around to display sections of the global environment. All manipulations to the function labels in the magnified environment (e.g., highlighting, as shown) is reflected in the global environment.

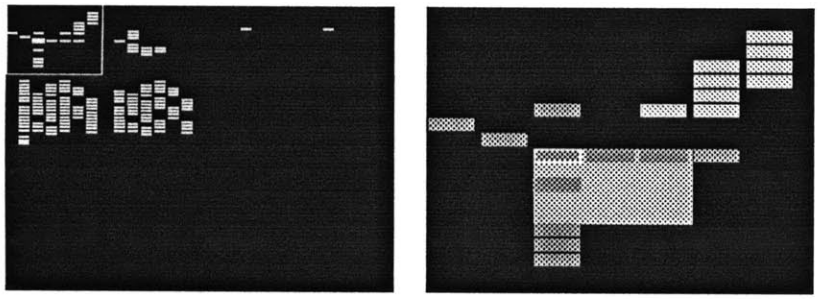


Figure 2.6c To inspect the code of a function, one only has to resize the function label. All changes to the code is done directly in the code-window.

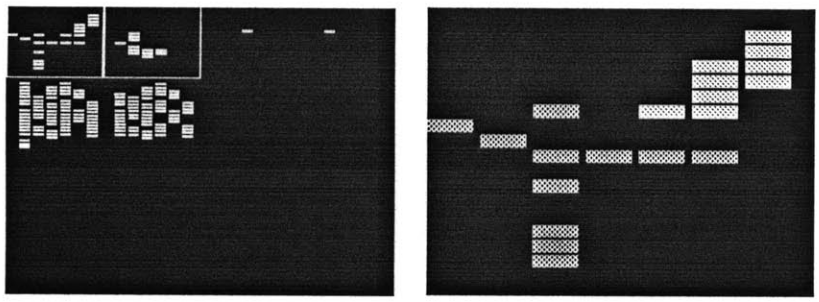


Figure 2.6d To compare functions that do not fall within the "scope" of a single scope-window, another scope-window could be created.

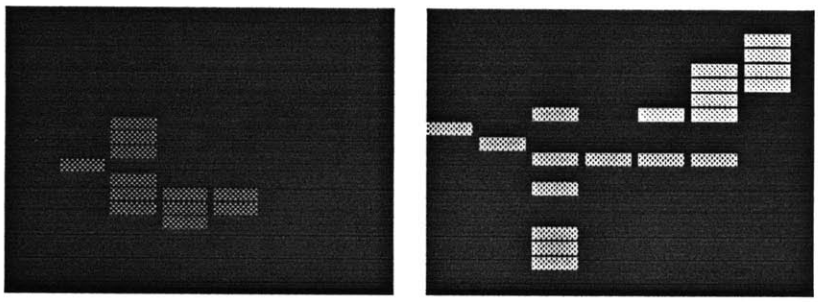


Figure 2.6e The global environment is temporarily used to display the second magnified environment as a result of the second scope-window. After comparison, the second magnified environment would be closed to reveal the global environment once again.

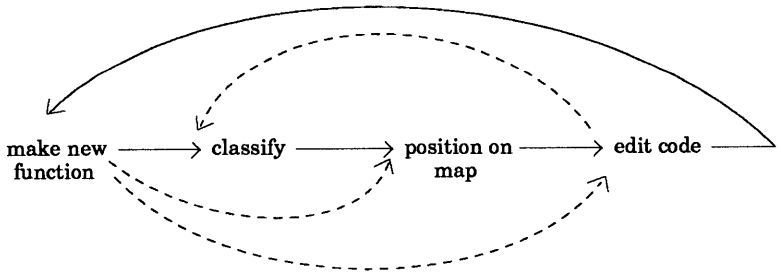
2.3 Building Up The Map-Program

2.3.1 Adding New Functions To The Map-Program

Nobody produces a perfect program on the first attempt. An interactive environment allows users to increase complexity with progressive additions and modifications. In traditional programming, the iterative process of additions and alterations is necessary before a program produces the anticipated result. Similarly, the construction of the map-program must go through the paces of expansion, rearrangement, and alteration. These activities reveal the similar nature between programming and design processes. A "conversation*" is set up between the designer and the design, and the process proceeds to and fro between the two, until a concensus is reached, which represents the solution. Throughout the construction process, the map acts as 'graphical recorder' - an instant visual feedback - explicitly displaying the programmer's intentions, and a mnemonic of his thinking process. Finally, the visual display of the map provides the continuity to tie the various components into a whole concept, hence a continuity of information. The process described above supports the contention that,

Programming = Designing = Elaboration of a sketch†

The map-based environment allows a user to successively adds new functions** to the map-program. The typical actions of a user is this basic loop:



The loop is not a rigid sequence of actions, as suggested by the dotted paths, which indicates alternate sequencing of actions. The user then proceeds to make a few more function-labels, and drags each map-function-label to a suitable location on the screen, so that they represent a sketchy outline of what the program is intended to perform.

* Donald A. Schon, *The Reflective Practitioner: How Professionals Think in Action. Basic Books*, New York.

† Intuition does not come to us unexpectedly as some people might imagine. Mostly, it comes gradually, in small portions, especially when we persevere with the work. Persistence and creative conditions encourage each other reciprocally. This is why interactivity is so important.

There is not a 'correct' approach, or solution, to a design-like task, such as programming. Personalization is the recognition of the existence of personal preference. However, preferences are not whimsical desires, but are based upon tangible qualities of the object.

** A menu selection *New Function* asks the user to name the new function being created. A function-label is created in which the user then has the option to change the default color, medium grey, to a coded color by selecting from a palatte.

Debugging has been left out of the loop to be dealt with separately.

Being in a dynamic environment, all displayed elements are potentially [manually] manipulatable. When an element is selected, it is in an activated state, indicated by highlighting (*luminance*, APPENDIX 5.3.1). Highlighting selectively emphasizes an element by drawing attention to the object. An activated, or selected element, can then be manipulated: moved, deleted, or altered by color.

2.3.1.1 An Example of Programming Using The Map-Program Environment

Let us look at an example as how we would proceed to build a map-program using the visual hierarchies mentioned in FIGURE 1.2. We know, for a start, that there would be five main sections to our program:

1) match
2) stream manipulation
3) forward-chain
4) rules
5) assertions

So, we makes five function labels, name them accordingly, and color code them with five distinct colors. The system acknowledges the five new functions by displaying them in the global-map. Remember that all manipulations to functions occur in the magnified environment, not in the global-map.

Now let us see how to implement *match**. We open the function-label *match*, and type directly into it the following pseudo-code. We adopt the problem reduction technique, dividing the general problem up into several more specific problems:

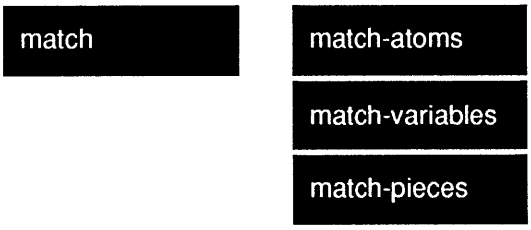
```
(defun match (p d &optional bindings) ;pattern and datum
    (cond
        ((and (atom p) (atom d)) ;both arguments are atoms
          ;;See if p and d are the same.
          ;;If so, return the value of 'bindings.'
          ;;Otherwise, return 'fail.'
          . . . )
        ((and (listp p) (eq '? (first p))) ;pattern is a variable
          ;;See if the pattern variable is known.
          ;;If it is, substitute its value and try again.
          ;;Otherwise, add new binding.
          . . . )
```

* From *LISP*, P.H.Winston and B.K.P.Horn, 3rd edition, Addison-Wesley.

Match compares a pattern expression with an ordinary expression to see if the ordinary expression is an instance of the pattern expression.

```
((and  (listp) (listp d))  ;both arguments are lists
   ;;See if the first parts match producing new bindings.
   ;;If they do not match, fail.
   ;;If they do match. try the rest parts using the resulting
     bindings.
   . . . )
(t  'fail)))
```

Now let us translate the comments into procedures. Seeing that there are three tests required, three more map-labels are made; given temporary names, *match-atoms*, *match-variables*, and *match-pieces*; color-coded in the same color as *match*, since they belong in the same catagory of functions; and arranged according to a self imposed convention that the functions on the right are usually called by those on the left. As shown below, the three new test-functions are clustered together because of their similar utility as tests for the arguments to *match*.



In exactly the same manner for *match*, we enter the code to the coding-window of *match-atoms*.

```
(defun  match-atoms  (p d bindings)
    ;;See p and d are the same:
    (if  (eql p d)
         ;;If so return the value of 'bindings'
         bindings
         ;;Otherwise, return 'fail.'
         'fail))
```

Having completed the function *match-atoms*, we can close its coding-window, clean up the defunct pseudo-code in *match*, and proceed to the next abstraction.

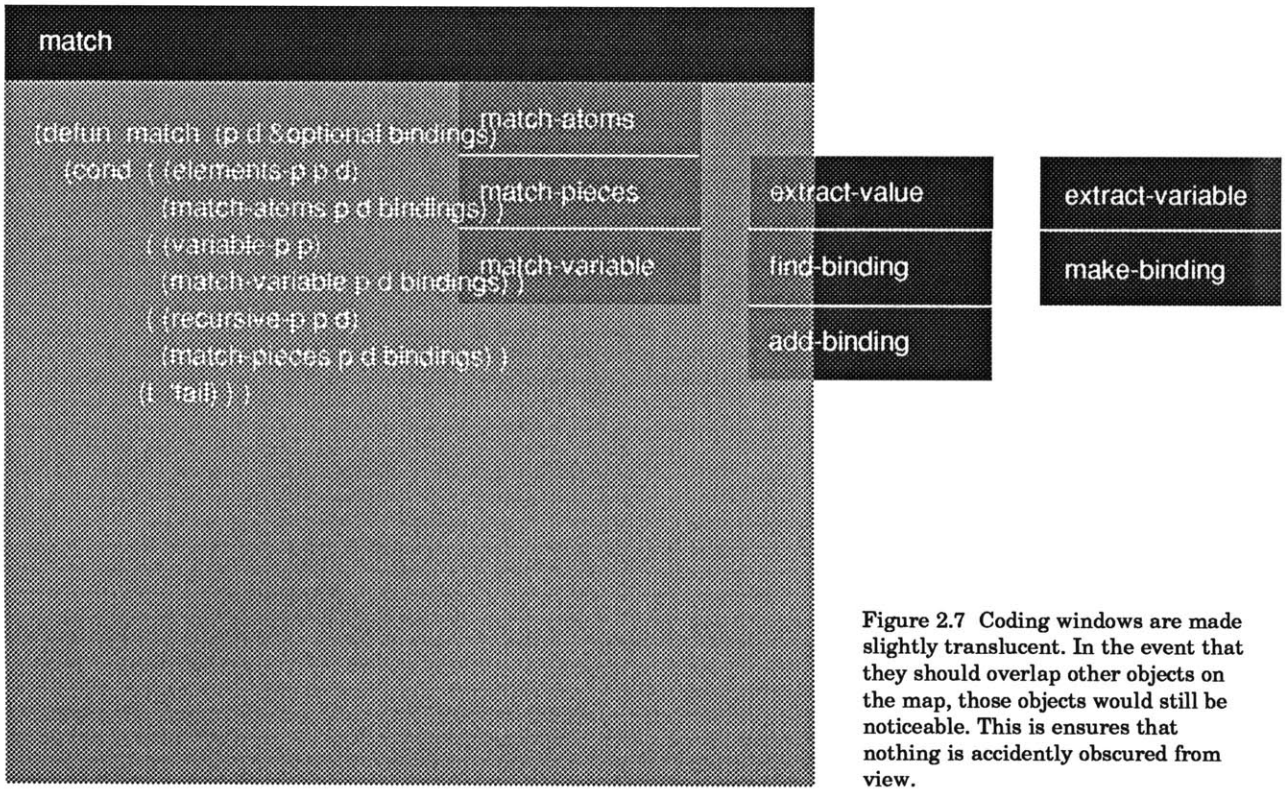


Figure 2.7 Coding windows are made slightly translucent. In the event that they should overlap other objects on the map, those objects would still be noticeable. This is ensures that nothing is accidently obscured from view.

In this manner, one function at a time, the program is built, much the same as traditional programming, except that the new environment has changed the way we look at the program. We now see it as a set of interacting discrete procedures, allowing for easy access between functions.

2.3.2 Accessing Information

The ease of access is defined as the speed of accessing a particular piece of information. The machine speed of retrieving information is negligible compared with the time taken for human acts of interaction with the interface. This time depends on several factors:

1) Mapping and memory: how intuitive are the environment and tools to use, and if substantial learning, which relies on memory, is required before a person can use the program.

2) Familiarity with the environment: knowing where things are and where to go to get at them.

3) Ease of operation: the number of actions or movements needed

to access the information; or if many different variations of access techniques (dialog) are employed.
4) Presentation: are the 'handles' (elements of interaction) to retrieve the information displayed in a convenient and logical manner, or are additional memory and effort required to access these handles.

2.3.3 The Index

To look up the location of an unfamiliar town in a print atlas, the index would be the place to start. The index provides a pointer to where the information could be found, in terms of a page and map number, the latitude, and longitude. However, a dynamic index, if available, would be able to display and highlight the exact location of the town. This is what we are attempting to accomplish for our map-program.

The purpose of an index is two-fold. First, it is an inventory of all the functions, as well as an ordering mechanism. Second, as mentioned, it directs the user to the location of the information - a search and show. As each new function is created, it is added to the index, and sorted by alphabetical order. Alternatively, the index could be sorted, first by color, and then aphabetically within the same color. Sorting by color is achieved by successive sorting the red, green, and blue component values of the color.

As just mentioned, having color-coding makes the search semantically laden. The user could perform a quick initial visual scan by color coding, following with an alphabetical search. It is not un- common that a user cannot remember the name of a function, but have a vague idea of its task. This is an improvement over the uni- dimensional 'grep' macro of UNIX, which performs searches strictly by character matching. Selecting a particular function-name in the index highlights the corresponding function-label on both the global-map and the magnified-map.

The index complements the hypertext technique. Hypertext allows a user to access information linked to another location, but it neglects the consideration of first getting to the starting location. While, in hypertext, there is often a need to return to the point of departure, the is no such need in an index. The danger of utilizing extensive hyper-linking capability, in general, is that of losing context. An additional facility to track the navigation in the hyper-world would be required, like the global-map.

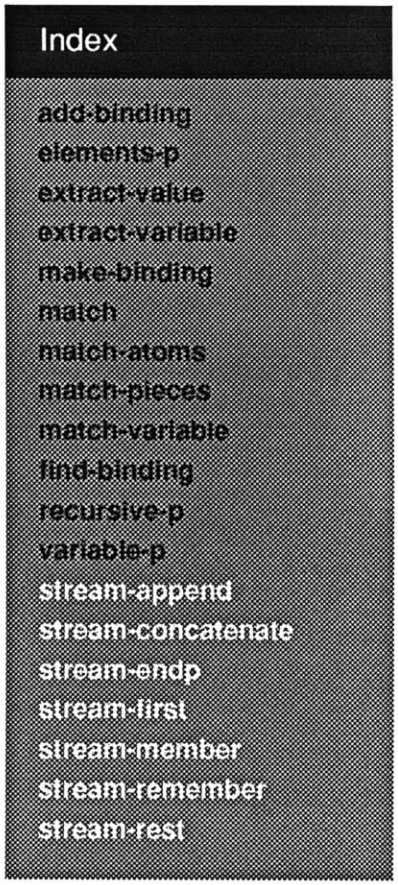


Figure 2.8 The scrollable index showing the sorting of the function names, first by color, and then by alphabetical order.

2.3.4 Moving Around

A large portion of programming consists of systematically moving from one function to another, tracing bugs; and elaborating, or adding functions to the program. In exactly the same manner as inputing the code for a new function, editing the code of a function is attained by opening the coding-window, doing the what is intended to the code, and closing when done.

For a selected map-function, clicking on the 'Calling' button in the menu would highlight, on both the global and magnified map, all the external* functions called by the map-function that are one level away. "One level" means that if function A calls B, and B calls C; B is one level away from A, and C is two levels away from A. Conversely, clicking on the "Called-by" option in the menu would highlight all the external functions that are one level away that call it. As with all interactions, the global map is updated to reflect every action on the magnified map.

The body-code of a function is seldom without calls to external functions. The routine to access the code of an external function, say *stream-first* from within the code of a function, say *forward-chain*, the procedures would be as follows:
1) Click on the text, *stream-first*, within the code of *forward-chain*; this would highlight the corresponding function labels *stream-first* on both the global and magnified environments;
2) Go to the function-label *stream-first* on the appropriate magnified environment and open the function.

A similar capability, in principle only, called "Meta-dot," is provided by EMACS. This utility locates the external function 'definition' called from the body code of a selected function. Different machines implement this utility in varying ways, but all in a non-perceptual manner. The drawback with this facility is that the user has to rely on his visual memory to maintain the continuity during the translocation from the calling function to that of the external function. It has been shown experimentally, that the short-term memory of humans to be inedequate, limited to about 7 ($\pm$2) items. In our map-environment, any jumping from one function to another is highlited in the global-map, minimizing the possibility of losing one's orientation. Furthermore, *hyper-* capabilities are essentially *goto* instructions, of BASIC. Without some form of assistance for maintaining context, the user could easily becomes confused, and lost in hyper-space.

---

* Since files do not exist in our programming environment, the term *external function* is used to refer to those functions called from within the body code of a function, not the external function in the C programming language sense.

Internally, the calls are managed by a two-dimensional array with the list of function names corresponding to the indices on both axes (columns and rows) of the array. All the slots of the array are initialized to zero. If function A calls function B, the slot at the intersection between the row A and column B, would be set to 1. To find out which functions are called by A, simply march along the row A, and pick out all the function (column) names whose intersect with row A contains the value 1.

It is debatable whether connecting lines are more appropriate in showing the calls between functions on the map. Undoubtedly, lines are the strongest elements in showing continuity (*continuity*, SECTION 4.2.2); but they also suggests a sequence of execution* which our map-program does not portray. To show the sequence and manner of execution in a program, a flow-chart representation would be needed. Our decision for the present is to use hiliting, and leave out the lines until a better use for them could be developed.

* When lines were used during demonstrations of the system, questions were always raised by viewers, as whether the lines did anything other than linking the calls, suggesting that more was expected of the lines.

2.4 Morphology of The Map-Program

The morphology of a program should reflect those features of the program being emphasized. It is desirable that the morphology of the map should have as close a resemblance to the user's cognitive map. This is termed the 'natural' morphology.

The global-map, with its equivalent magnified sections, is an isomorphic/analogical representation (APPENDIX 5.1.4.1) of the user's mental model of the information's (function's) organization in the map-program. The arrangement of the function-labels by the user during the process of building up the map-program, could then be in accordance with some criteria imposed by the user.

For example, within functions of the same color-code, we could further cluster functions that are operationally similar (FIGURE 2.4). We could also arrange the function-labels such that from the stand-point of a selected function, those to its right would be functions that it would call, and those on its left would call it. Finally, groups of functions with similar color would be positioned on the global map in a strategic relationship to each other, creating a shape that correlates well with a user's mental picture of the map-program. There are no hard rules in which to regulate the arrangement of functions in the map-program.

From the description of the interaction between the user and functions in the map-program environment, sequential access of information has not played a significant role. This demonstrates that access of information and the execution of the information are not the same. However, the representational emphasis of the map-program is organization, rather than the sequence of execution. If the emphasis was on tracing the execution of the program, the morphology of the map-program would be different, and might lean towards that of a flow-chart.

2.5 Debugging

Debugging is a complicated process. While it would not be possible to address all aspects of debugging, we shall look at those aspects of debugging that could be improved by using visual design principles and organization.

Debugging is essentially a process of isolation, whereby the location of the cause of the problem is systematically narrowed, as illustrated in FIGURE 2.9. The process of isolation when presented visually, explicitly defines the portions of the program that have been 'filtered' and directs our attention to those that have not. Therefore, a spatially oriented environment should facilitate the process of debugging.

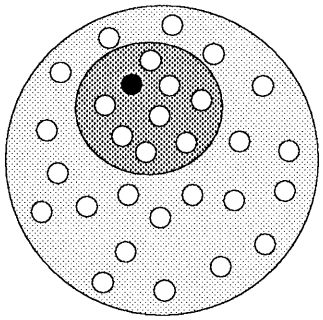


Figure 2.9 Successive isolation of sought after information.

Stepping* is the technique of sequentially evaluating each expression in a function, or each function in a program. Initially, in a coarse step -through of the program, each function on the global-map and its magnified portion, is highlighted as it is evaluated. If a syntactic bug is encountered causing a halt in the execution of the program, the exact location of the bug would be the highlighted map-function label, frozen at the time of the interruption of the program. We can then open the coding window of the bugged function, and "Lo!" the very bug would be highlighted† before our eyes.

LISP allows the evaluation of individual functions during the process of programming. As with stepping through the program at the map level, stepping through a function involves evaluating, and thence, highlighting each individual expressions in turn. If a bug is encountered that caused the execution of the program to halt, the highlighting stops at the location where the bug has been encountered, immediately drawing attention to any interuptions in the flow of control. In the example below, an unbalanced parentheses halted the evaluation of the function. The unbalanced parentheses is highlighted.

```
(defun match (p d &optional bindings)
  (cond (  (elements-p p d)
           (match-atoms p d bindings)
        ( (variable-p p)
          (match-variable p d bindings) )
        ( (recursive-p p d)
          (match-pieces p d bindings) )
        (t 'fail) ) )
```

* Henry Lieberman, *Steps Toward Better Debugging Tools For LISP.* 1984 ACM Symposium on LISP and Functional Programming.

† The problem becomes very different when bugs are not due to syntactic errors but to incorrect variables assignments. The error does not cause a halt in the execution of the program, but simply produces unexpected results. It would then be necessary to have a mechanism that would allow the inspection of the variables at every point of the execution of the program. Unfortunately, such a debugger has yet to be developed, but would certainly deserve future work.

2.6 Visual Details

2.6.1 Visual Refinements

In this section, we shall discuss the suitability of a textual representation of functions in terms of:
1) readability* of the code, and
2) the organization and presentation of the code.

Readability is as much a function of costs, as is the syntax of the programming language, or even the skill of the programmer. Having a high quality graphical display is a constraint of hardware costs. The technology is available. The map-program environment uses a set of anti-aliased and proportionately spaced fonts on a high resolution (1280 x 1024 pixels) Hewlett-Packard 24-bit Renaissance frame-buffer, capable of supporting true transparency. Anti-aliasing smoothes out any jaggies that are still visible, despite the high resolution display, and allows a smaller font size to be legible, saving valuable screen space. Anti-aliased fonts also allow for more comfortable reading at a healthy distance from the screen.

The syntactic form of a language can affect its readability. One characteristic of LISP that is perculiar is that its instructions are expressed as a composition of nested functions, rather than as a series of sequential steps, as is the case with most other conventional languages. Also, in LISP, the desired operations are expressed in the form of a single complex function that is composed of simpler functions. This is a consequent of the unique way that LISP treats both its functions and data undifferentiated. Hence, in a LISP function, the deeply nested sub-routines are sometimes read backwards, which is contrary to our being accustomed to reading from left to right.

* Readability is defined as the ease of reading due to the physical characteristics, not the semantics, of the information. Readibility includes legibility, which is the capability of information being read.

2.6.2 Grids

Along with the refinement of visual characteristics, is the quest for maintaining spatial order. A visually well ordered environment improves its effectiveness by minimizing information access time. We extend the tried and true technique of the grid, into the dynamic medium of the electronic display. There are appreciable differences between designing a grid for a static medium and a dynamic medium. In the static medium, an item, once placed in a particular location cannot be moved. The formal relationships between the elements are maintained permanently, whereas in a

dynamic medium, the item is still movable and resizable. More importantly, in a dynamic medium, both the format and the display items are active participants in layout process. A nominal amount of "intelligence" in the form of constraints could be applied to both the format and the visual items to produce the desired results*. Three commonly used techniques of designing with constraints that would be applied to the grid are: *domain constraint*, *snap-resize* and *snap-move*.

An invisible grid made up of modules of a certain size, for example, 128 x 128 pixels, exists over the display. The module could be divided into sub-modules, for example, there could be eight sub-module of 128 x 16 pixels stacked up to make a module. With snap-resize, any image when resized, would automatically be constrained to a multiple of the sub-modular size. This technique is used in the resizing of the code-windows of the map-functions. Similarly, when an item is moved on the display, its resting position is snapped to the nearest invisible grid line, automatically aligning the item. This technique is used in all movement and placement of function labels. With domain constraint, an item could be constrained territorially to within a prescribed area, for example, within an invisible column. The use of this technique is not as apparent as the other two. Functions within a magnified environment are constrained to move only within the environment.

* The analogy of an ant walking on a pebbled beach by Herbert A. Simon in his book, *The Science of the Artificial*.

"Consider the path an ant makes on a pebbled beach. The path seems complicated. The ant probes, double back, circumnavigates and zig-zags. But these actions are not deep and mysterious manifestations of intellectual power. Rather, the ant is a simple solver operating on a complicated beach with alot of pebbles to get around..."

3.0 Conclusions

We have to ask the question as to why is the "single-file" map-program environment possible now and not before. It would not be entirely accurate to say that it would not have been possible before, but it would be true to say that had it been implemented a few years earlier, it probably would have been rejected, due to the fact that the crude level of graphics and visual capabilities of the earlier machines would have made the idea look too cumbersome for serious consideration.

This project is a timely confluence of opportunities, and a natural marriage of visual design knowledge and new technology. Directories and files are non-graphical methods of classification and organization. The fact that the visual hierarchies of the map-program would not be achievable without a sophisticated visual environment, in fact, is totally dependent on the availability of a sophisticated and interactive display.

In demonstrations of the map-program prototype, viewers have reacted positively to the feasibility of the concept. Hopefully, it would be a beginning for further development beyond the domain dealt in this thesis. Good prospects lie in developing a visual interaction model, which reflects the sequential nature of code execution, something which has not been dealt with in this thesis. The question of multiple users accessing and manipulating the same database of information has always been a problem in computing. It should be interesting to find out, if a visually enhanced environment would reveal any new angles on the problem.

The long term goal could be one of visualizing new interaction models of programming conform to the way humans "design" things. We have shown that programming and designing share very similar characteristics of iterative assessment and improvement of the work in progress, and this could be a strong basis in which future programming environment models are built on.

4.0 Appendix A : Visual Principles For Effective Use Of Dynamic Information

---

"In popular usage, the term information refers to facts and opinions provided, and received during the course of daily life. As a person uses such facts and opinions, he generates information of his own, some of which is communicated to others."

[Encyclopedia Britannica]

4.1 Representing Information

A model, when imposed on incoming data, enables large quantities to be dealt with as a single coherent, or coordinated unit. A representation is the process of converting a given problem into another problem that has a known solution. In representing information that is not inherently visual, such as sound and quantity, our fore-fathers have expended much effort to devise musical scales, and numbers. We know today that the concept of 'zero', or 'nothing' was particularly difficult for them to grasp.

Acoustical information could be given visual dimensions. It is simply a matter of finding an appropriate representation for the problem at hand. A rich source of reference material exists in dadaist and futurist works. David Small's thesis (MIT Media Laboratory, MSVS '90), Expressive Typography, examines the relationship between sound and typography.

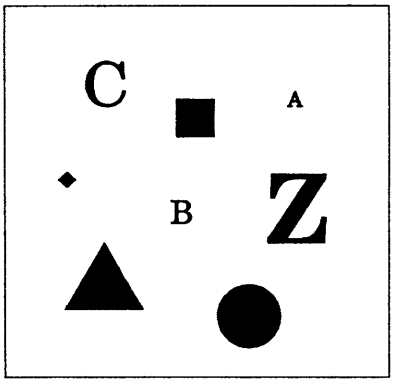
4.2 The Multi-dimensional Nature of Information

Information is dimensional, with dimensions such as formal attributes (color, shape, size, acuity, etc.), magnitude, and duration. '5' has both magnitude, and lineal spatial dimension, in the sense that it occupies a positioned between '4' and '6'. The task of the information designer is to manipulate these dimensional attributes for better communication. Generally speaking, a well structured and

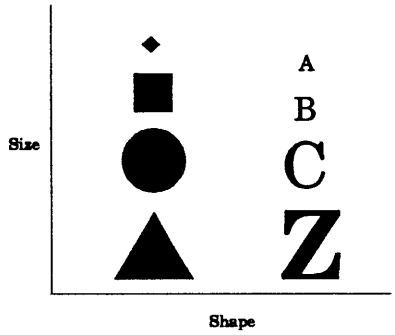
organized environment facilitates access (and navigation) of information, by presenting information in a systematic and logical manner.

4.2.1 The Spatial Dimension: Spatial Disposition In The X, Y, and Z axes

Spatial disposition is defined as the relative position and spatial arrangement of one or more objects in an environment. Spatial disposition and motion in the x-y plane is straight forward. On the z-axis, the issue becomes less obvious. Transparent overlapping of graphic elements is routinely employed in the print medium to create an illusion of depth (z-space), as well as to maintain unity of composition. In computer displays, however, it is a recent phenomenon. Transparent overlapping of visual elements allow one visual element to be visible through another, or more layers of other visual elements. Ambiguity might exist as to which layer is on top, or behind which other layer(s). Thus, while they are not really 3D, they still exhibit rudimentary depth perception, and is real, hence the term 2½ D. Without transparency, information that is obscured would not be visible, unless it is "brought to the screen's surface." It exists only in memory, an imaginary space, but accessible, being linked and retrievable. This technique is employed extensively in hypermedia systems.

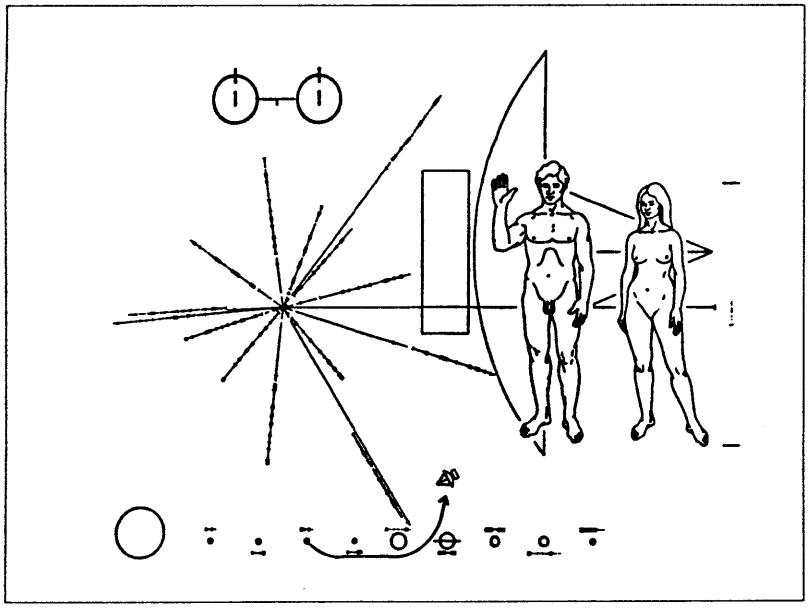


(a)



(b)

Figure 4.1 a and b. The first step in procesing raw information is to classify them according to some dimensional criteria. Figure 4.1c, (below left), shows that much of how we describe our world also depends on classification. This diagram was mounted on the side of Pioneer 10 and meant to be read by intelligent aliens in space. This diagram was the brainchild of the American astrophysicist Carl Sagan.



(c)

4.2.1.1 Spatial Disposition Can Provide A Big Leap In Comprehension Of Information

Compare the two representations of the Boston subway, FIGURE 4.3 and FIGURE 4.4. The tremendous improvement in comprehension of the data (FIGURE 4.4) rests solely on the relative disposition of the information on the map. In addition, color coding would improve considerably the ability to distinguish the different groups of information.

The example above, demonstrates convincingly that spatial disposition is a powerful cue in the comprehension of information. By having the information laid out in their proper relationships, the structure of the information has been made explicit beyond the capacity of textual description.

In a dynamic environment, information is not confined to a fixed location, but is free to move around, constantly changing its spatial relationships to other pieces of information in the environment. If we liken textual programs to the textual representation of subway information in FIGURE 4.3, we can sense how spatial disposition could be relevant to programming.

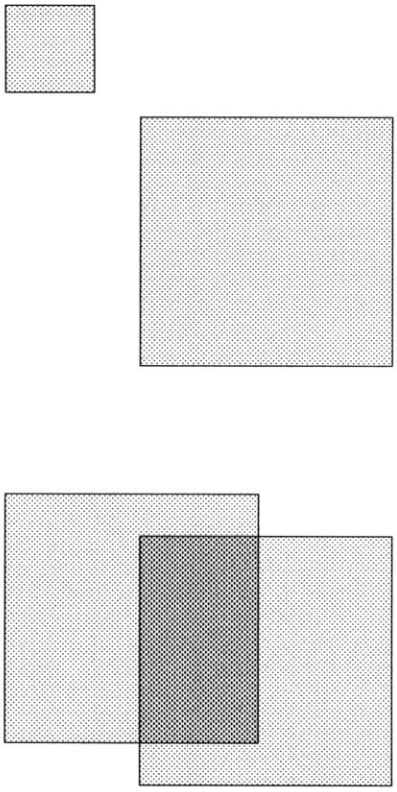


Figure 4.2. In the top example, size is used to create the *illusion* of depth. In the bottom example, ambiguity exists as to which plane is on top.

The method of slipping planes, each behind the other, or interpentrating planes, had become standard for spatial abstraction by the 19th century, in painting as well as in architecture. Pablo Picasso in his mural of the bombing of Guernica, said that he had attempted to render "the inside and outside of a room simultaneously".

4.2.2 Continuity (Unity, Context).

Context is the connecting of the parts, or the relationship of the parts with respect to the whole, like the beads in a necklace. Continuity is the measure of the 'smoothness' of flow (as oppposed to interruption) through the parts. Its function in communication is, therefore, a vital one. When skipping from one 'chunk' of information to another, a continuity of information between the two must be maintained to 'make sense' in terms of the overall picture; that is, the relationship of individual elements within the context of the overall objective.

In FIGURE 4.5a and b, the black dots, lead the eye to travel along a path; do the grey dots. The similarity of color provides the element for continuity. In FIGURE 4.5c, lines are added to make explicit the continuity. The eye now follows the line-linked dots even though their colors are not the same. Lines provide explicit associations. There are several ways to achieve visual continuity of information. They include proximity, and likeness (shape, color, size, etc.), but the strongest of which remains the direct joining of the parts with a line.

Red Line: Alewife, Davis, Porter, Harvard, Central, Kendall/MIT, Charles/ MGH, Park Street [1], Downtown Crossing [2], South Station, Broadway, Andrew, North Quincy, Wollaston, Quincy Center, Quincy Adams, Braintree.

Green Line: Lechmere, Science Park, North Station, Haymarket, Goverment Center [3], Park Street [1], Boylston, Arlington, Copley, Auditorium, Kenmore, Boston Univ., Boston College.

Orange Line: Oak Grove, Malden, Wellington, Sullivan Square, Community College, North Station, Haymarket, State [4], Downtown Crossing [2], China-town, NE Medical Center, Back Bay/South End, Massachusetts Ave., Ruggles, Roxbury Crossing, Jackson Square, Stony Brook, Green Street, Forest Hill.

Blue Line: Wonderland Revere beach, Beachmont, Suffolk Downs, Orient Heights, Wood island, Airport, Maverick, Aquarium, State [4], Government Center [3], Bowdoin.

[1] Red-Green lines interchange.
[2] Red-Orange lines interchange.
[3] Blue-Green lines interchange.
[4] Blue-Orange lines interchange.

Figure 4.3. The simplified routes of the Boston Rapid Transit Lines.

Figure 4.4. The spatial disposition of the Boston Rapid Transit Lines. Artwork courtesy of MBTA.

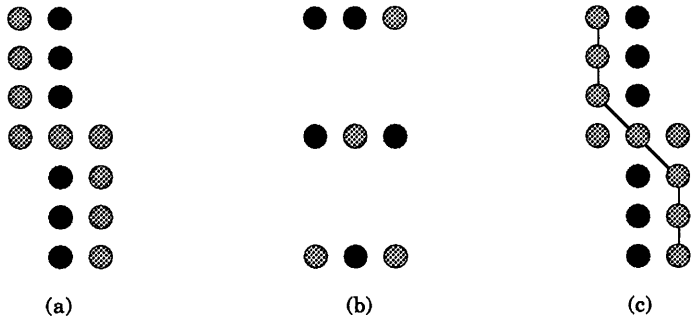


Figure 4.5a, b and c. In (a), continuity is maintained by sameness of color. In (b), proximity is used to maintain continuity vertically, but notice that color still exert strong horizontal influence, indicating that color is a very strong grouping parameter. In (c), explicit linking by lines is used to overide continuity by sameness of color or continuance.

Comparisons are often made between separate chunks of information which are mutually occluded in view, like on different pages of a book. A static medium forces the user to rely on his visual memory (flipping back and forth quickly between the two sections of the book). The continuity would have to be pieced together in the mind. In a dynamic medium, these scattered chunks of information could be brought together for review simultaneously. In programming, where this problem is prevalent, multiple windowing alleviates some of the difficulty by allowing the multiple windows to be viewed simultaneously, maintaining local continuity, but makes no attempt in solving the problem of context in the overall program.

4.2.3 Locality of Information

This principle should not be confused with spatial disposition of information. Locality of information dictates that information describing an object should be at the same location as the object it describes. This is only rational. A person should not have to see an object in one place, and look for its descriptive information some where else. What we are affecting here is the least possible energy expenditure to accomplish a task - efficiency. Things which are related to each other usually share certain common attributes, such as location, color, shape, etc.* to facilitate access.

Locality of information is a simple principle whose full potential could only be exploited in a dynamic environment. With print, the desire to achieve locality of information can easily lead to clutter as shown in FIGURE 4.6, making the access of information tiresome. Clutter is a problem on all print maps, no matter how well designed they are. To overcome this problem, techniques were devised to spread out the information, like pointers, legends, and even multiple maps, with each map showing different catagory of information. But this raises another question. Multiple maps make comparison of data difficult, if the data reside on different maps.

* Information that are related would be located close together so as to minimize the physical effort of accessing the items. The same principle is utilized in the layout of factory-floors and homes (e.g., the kitchen).

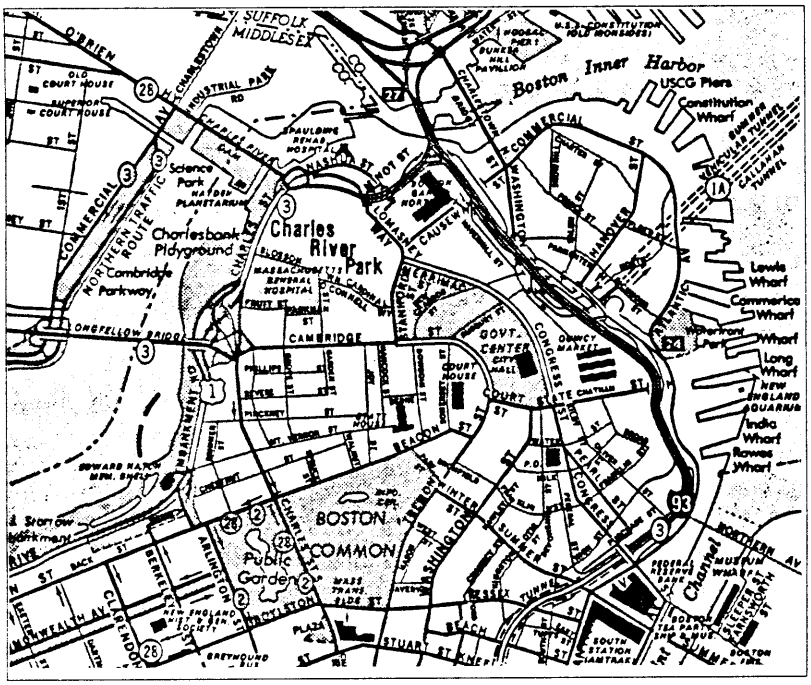


Figure 4.6. In order to achieve locality of information in a static medium clutter can result. In the following section, we shall see how *selective emphasis* of information could enable locality of information to be achieved without clutter.

4.2.4 Selective Emphasis of Information

The concept of *selective utility* (time-sharing)of information, means is that an object or event has periodic utility and dormancy in a system. Given an over-abundance of information (such as on a map), not all the information is needed at the same time. At any particular instance only a relatively small portion of the total amount of information is required. A dynamic medium allows information that is not required at that moment to be hidden away (or partially hidden, using translucency; and blurring), giving that information that is relevant, and on display, greater prominance. After use, the information is removed until the next time it is called upon. The period for which a piece of information is needed depends on 1) its utility to the the user at a particular time, and 2) its sequencing (scripting) in conjunction with other pieces of information. Hyper-media, is a good demonstration of this principle of selective utility. Related information is linked together to be retrieved as needed. Even if superflous information do not constitute 'clutter', there is still no reason for it to lurk around. If it exists, it would distract from other information on display.

Selective emphasis can be utilized in other ways. Another is the *selective amplification* of needed information, without hiding the other information present. For example, in a tangled web of linked nodes, a selected link could still be easily distinguished by highlighting it. The alternative is *de-emphasis* by dropping out the

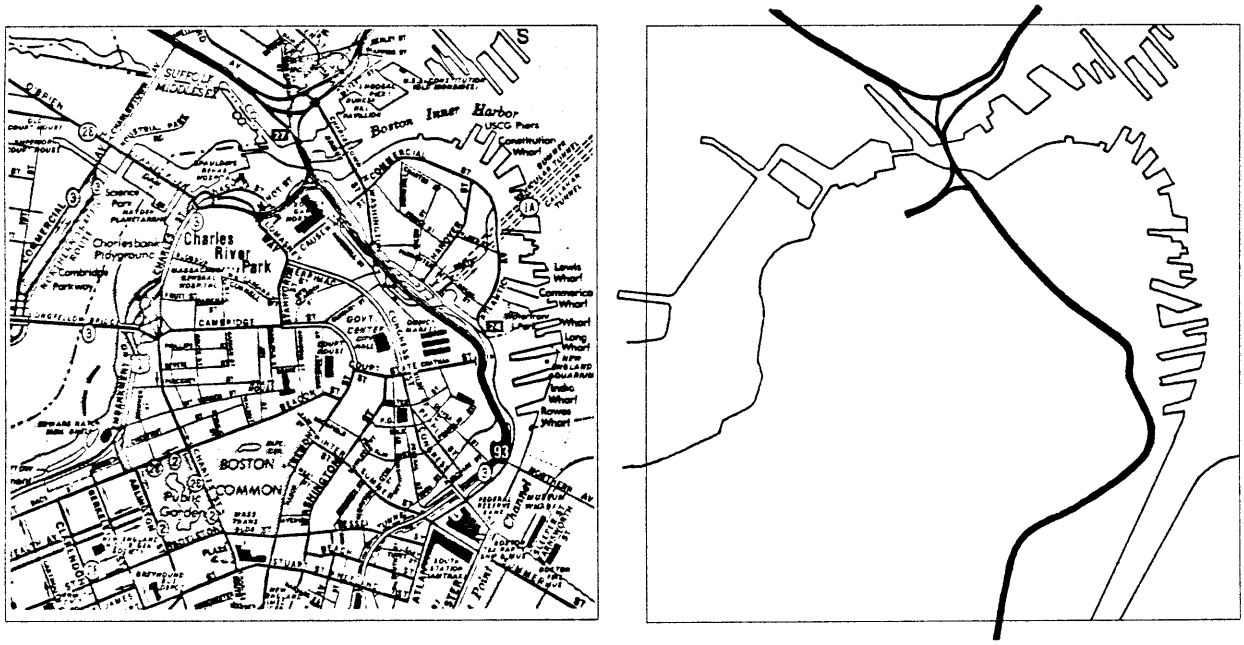
non-essential information completely, or partially. Because important features can be amplified selectively, a dynamic medium can tolerate a higher degree of disorder in the presentation of information. However, this statement must not be taken as an excuse for tardy design. Dynamics allows us to make the right relationship obvious.

The third use of selective emphasis is *cropping* or *framing*. This technique is usually used in conjunction with scaling. The camera is the best example that illustrate this technique. Simply by framing on a particular object (or part of an object), gives the selected object 'center-stage' importance. Cropping can produce unintentional side effects. To give an exaggerated example, a picture of an eagle in flight, and a 'close-up' of its talons ellicit very different emotional reactions.

Related to the idea of visual scaling is that of *content scaling*. Alternate names include *information resolution*. Content scaling is different from selective emphasis by scaling, or visual scaling, which is based on visual information in the same visual representation, that is, already on display. Although visual scaling could have semantic side-effects, as illustrated above, content scaling is the deliberate effort to emphasize, or de-emphasize the amount of information in a representation, hence, its alternate name, information resolution. Information from different sources could be used, so long as they act together in enlarging the understanding of the perceived information. This concept is very well illustrated by the book *Powers of Ten*. *

*Eames, Charles, Morrison Philip and Morrison, Phylis. Scientific American Library.

Figure 4.7. Selective emphasis by hiding irrelevant information.



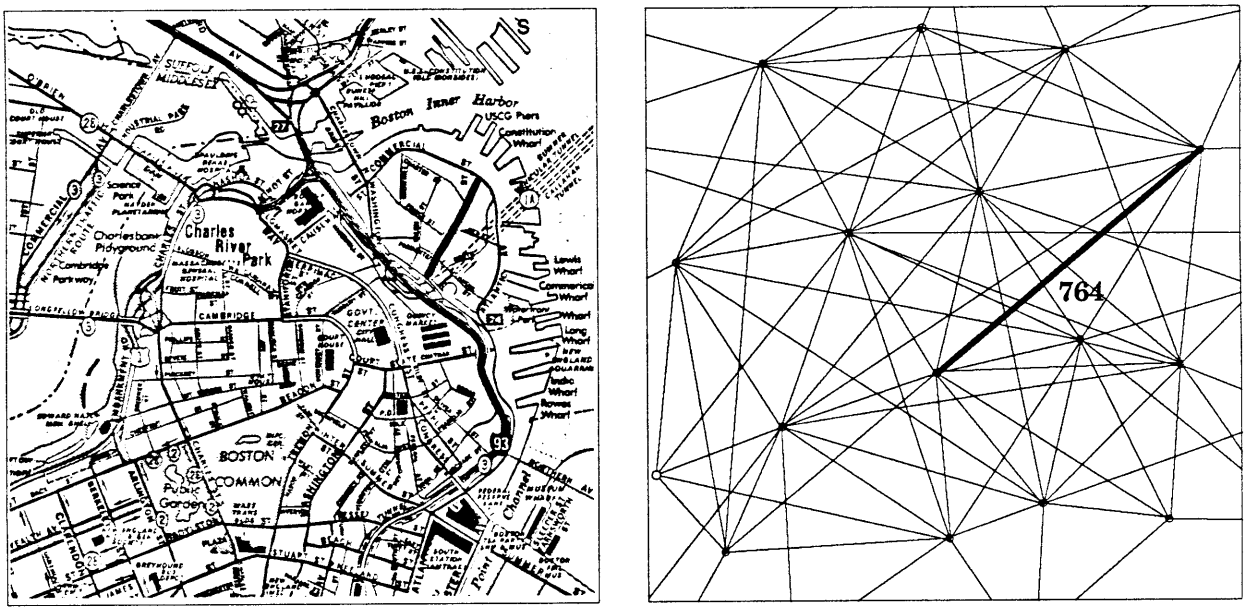


Figure 4.8 Selective emphasis by selective amplification using visual characteristics (e.g., color, luminance, size, weight, etc.).

4.3 Formal Considerations - Visual Quality.

Formal dimensions of information could be divided into two parts. The first, which deals with the information itself (visual dimensions), includes color/grey-scale, size/scale, shape, acuity, and visual texture. The second, which deals with the interaction between the information and the environment housing it (spatial dimensions), and amongst the different pieces of information, includes: balance, gestalt relationships, unity, and order.

With regards to the formal properties of the information itself, there exist substantial applicable knowledge from research on color, size, proportion, shape, and acuity, of traffic signage that could act as valuable pointers to how we deal with dynamic information. In traffic signage, the viewer is moving, and the information is stationary. In dynamic information displays, the relative motion between the viewer and information is reversed: the information is moving, while the the observer stationary. A sampling of the findings from traffic signage research are listed in APPENDIX 5.3.

4.3.1 Pictures are not always better than words.

The widespread notion held by the ardent advocates of visual programming that pictures are more powerful than words as a means of communication is debatable. No one has yet been

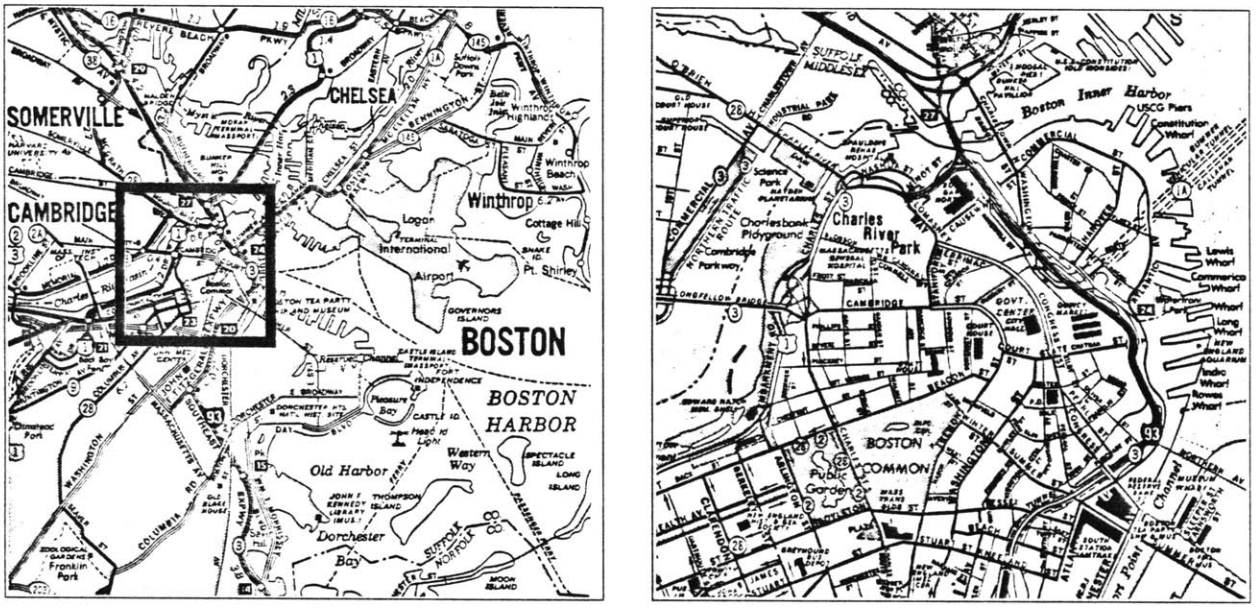


Figure 4.9 Selective emphasis by cropping and scaling.

able to come up with a satisfactory visual representation of such fundamental concepts as "IN" and "OUT", let alone a notion, like "beauty", which is open to personal qualification. Pictures may be more suitable for describing physical details. Words are more economical for describing abstract actions and concepts, where the meaning of the words are commonly or culturally understood, for example, "IN". Text and images complement each other, and it would be prudent not to favour one to the exclusion of the other.

4.4 Forms of Graphical Representation

At the highest level, visual information are differentiated into two kinds of representations, text and graphics. The boundary between them is sometimes hard to discern, as exhibited in some works of typography.

All information could be depicted by one of four types of graphical representation: isomorphic/analogical representation, matrix, relational-net, and tree-diagram. These are described in APPENDIX 5.1.4. One shortcoming common to all four representations is their restriction as two-dimensional representations. Unfortunately, a full treatment of graphical representation is beyond the scope of this thesis.

An appropriately chosen representation serves the purpose of converting a given problem into another problem that has a known solution. A good representation makes clear the important

features of the information. Other roles of representation are cited by Fischler and Firschein*:

1) Interpretation. Visual information can be interpreted by comparing the sensual visual data with stored description of objects.

2) Organization. A representation may allow us to organize information so that similarities and differences between objects and events are more readily identified. Plotting two sets of data on the same graph will visually show similarities and differences.

3) Questioning. Representations lead us to ask questions about events. We are thus guided to revisions in our models, the generation of a set of alternative models, or further attempts at data gathering from our surroundings.

4) Prediction. Representations allow us to predict events that will result from actions; for example, a mathematical model of a rocket enables us to predict the motion of the rocket.

5) Deduction. Certain representations can be used to make new knowledge explicit by allowing deductions to be performed on original knowledge. For example, maps are for navigation; we need to navigate information, and hence we can deduce that maps would be a suitable metaphor to use.

* Martin A. Fischler and Oscar Firschein, Intelligence: The Eye, the Brain, and the Computer. Addison-Wesley, 1987.

4.4.1 Navigating Through Information: The 'Map' Metaphor

A map is a visual notation of the environment, and was invented for the purpose of navigation and identifying locations. Context is crucial. Looking at the stylized subway map (FIGURE 4.4), the relationship of each station to the overall subway system is clearly presented. Normally, a stranger to Boston would not have too much difficulty "navigating" his way from, say, MIT on the Red Line, to Boston College on the Green Line. FIGURE 4.4 is a gross abstraction of the actual route (FIGURE 4.10). Information that is not essential, such as the actual distance between two stations is left out in the abstracted version. Traditional maps, because of their static nature, are often crammed with more information than is needed to satisfy a typical use. The density of information on print maps is a curse of its static nature, not necessary a welcomed challenge to good design.

There are many kinds of maps. Forget for the moment as to how a map ought to look, and analyse what constitute a "map". That is, what is the function of a map, or why it is needed?

1) Maps show the relative position of nodes (towns, stations, etc.) to each other.

2) Maps show how to get from one node to another (routes, links),

by what means, and the duration it takes.
3) Maps show the conditions and features of the environment in which the nodes reside, and the available resources in that environment. The site of these nodes are usually a direct consequence of the features of the environment.
4) Maps often show the relative importance of the nodes (scalar parameters).

4.5 Work of Similar Nature

Pretty-printing, uses one technique: the indentation of sub-procedures into nested blocks. For that, it does a remarkable job, producing a big improvement in readibility. However, more is needed. Knuth describes what he calls literate programming, the idea that program text and a verbal description of the program explaining it to a human reader should be integrated. He describes a compiler which produces both executable code and hardcopy documentation from a single integrated document.

Aaron Marcus's 'book' metaphor extends on pretty-printing, by making the textual matter of programs closer in appearance to the accepted standards of book publishing*. Driving a laser printer equipped with a set of fonts, a visual compiler takes a C program as an input, and produces a greatly enhanced presentation of the same program as an output (see FIGURE 4.11). But, it does not stop the question from arising as why is it necessary to produce hard copies *if* the on-screen visual quality is as good as, or approaching that of print. Also, the quality of the laser printer far surpasses the then available screen resolution, creating a disconcerting discrepancy when correlating between on-screen information and the print-out information.

Henry Lieberman† seeks to amalgamate the visual hierarchies as exhibited by the headlines and titles of the newspaper with programs. In that respect, the analogy is well used, and several poignant issues were raised. However, one might add that the layout of the newspaper operates on very different criteria than a program. For one, the nature of space is very different. It is static! Economics play a deciding role in the newspaper. In the attempt to pack in as many stories as possible, almost all the mutually independent stories that begin on the first page have to be continued on another page, creating discontinuities that would be intolerable for programs on screen.

* Baecker, Ronald and Marcus, Aaron, *On Enhncing the Interface to the Source Code of Computer Programs.*

† Lieberman, Henry, *Layout as Interface: From Newspaper Design to Program Design.* Unpublished paper.

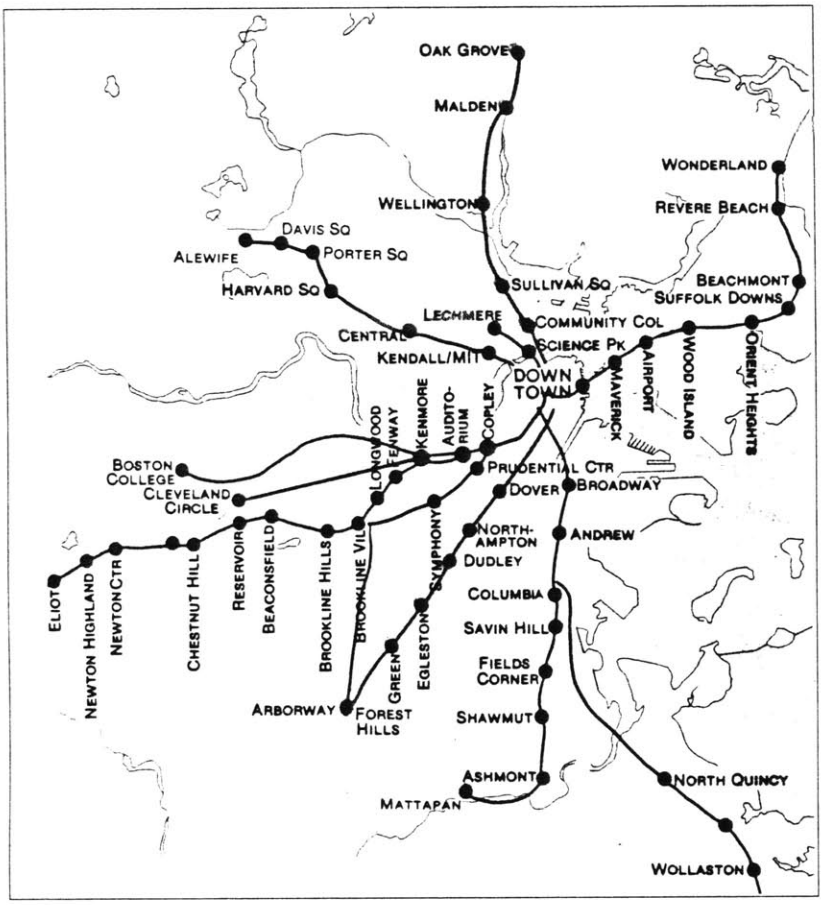


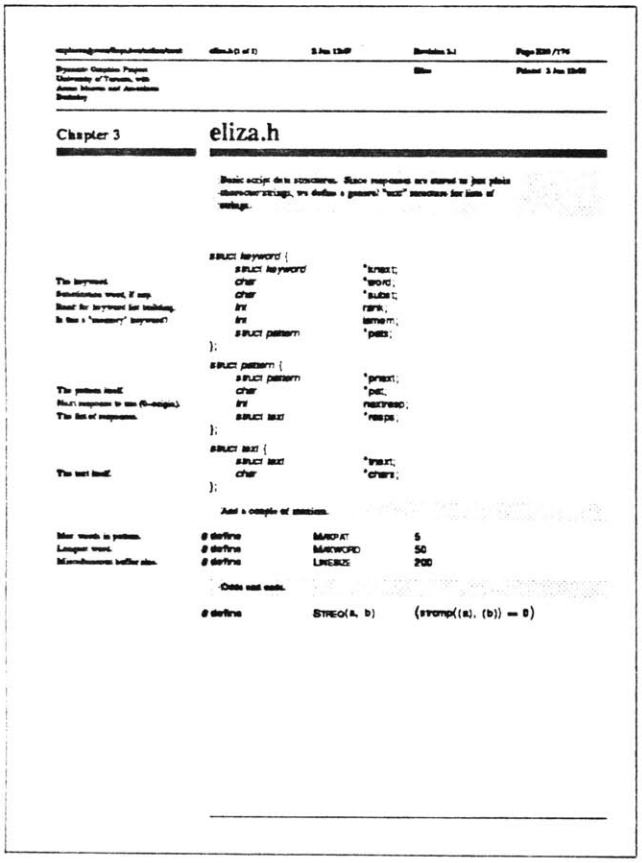
Figure 4.10. The actual routes of the Boston Rapid Transit Lines.

The surface area of the monitor is small, but with scrolling (scrolling seems to cause the same problems as flipping pages back and forth), becomes 'virtually' limitless in size (hardware limitations), resulting in very different design criteria. To aggravate the situation of screen surface area, the size of on-screen text, even at high resolution needs to be at least twice the size of newspaper texts for comfortable reading over prolonged periods of time, because the viewing distance between the reader and the screen must be maintained at a 'healthy' distance. Finally, the visual 'character' of a newspaper is something to abhor rather than emulated. It is a compromise with economics and the medium. The visual density of information, in which readers have no choice to select, produce visual clutter, and a visual form that is not directed to the end-users' convenience.

One should be aware of the dangers in using analogies and metaphors. While analogies may have their usefulness in triggering ideas, analogies do not usually address the same specific problem that is to be solved. For every idea that is adaptable from a metaphor, there is another, or more, that is misleading. Most misleading

of all, is to think that one could take only those ideas that are applicable, and leave the unapplicable ones behind. Each problem has its own unique demands on a solution.

This thesis opposes the point of view behind the setting of "good" typographic rules. Rules, although sometimes necessary, are easy solutions to problems, and often become unnatural constraints. It is more helpful to point out the issues that could lead to the desired goal, while leaving the decisions to the individual users to satisfy their paticular stylistic preferences. It does not mean that standardization should be avoided at all cost. Standardization, quite often, provide the only solution to a problem. Traffic is a good example. However, standardization should be used only when all other efforts in solving a problem have failed, not as a first option.



Chapter 3 eliza.h

```
struct keyword {
    struct keyword  *knext;
    char            *word,
    char            *subst;
    int             rank;
    int             lemem;
    struct pattern  *pats;
};
struct pattern {
    struct pattern  *pnext;
    char            *pat;
    int             nextresp;
    struct text     *resps;
};
struct text {
    struct text     *tnext;
    char            *chars;
};

#define     MAXPAT      5
#define     MAXWORD     50
#define     LINESIZE    200

#define     STREQ(a, b)     (strcmp((a), (b)) == 0)
```

Figure 4.11 Example print-out of Baecker and Marcus's "book metaphor" program code. Reduction seen here is 40% of original.

5.0 Appendix B: Principles of Information Design

The task of visual information design could be summarized as: to make information visual, and easily comprehensible. The following sub-topics are in no way discrete, but are inter-related; and identify the major issues that need to be considered in the design process.

1) Representation
2) Interaction
3) Formal quality
4) Technology
5) Economics

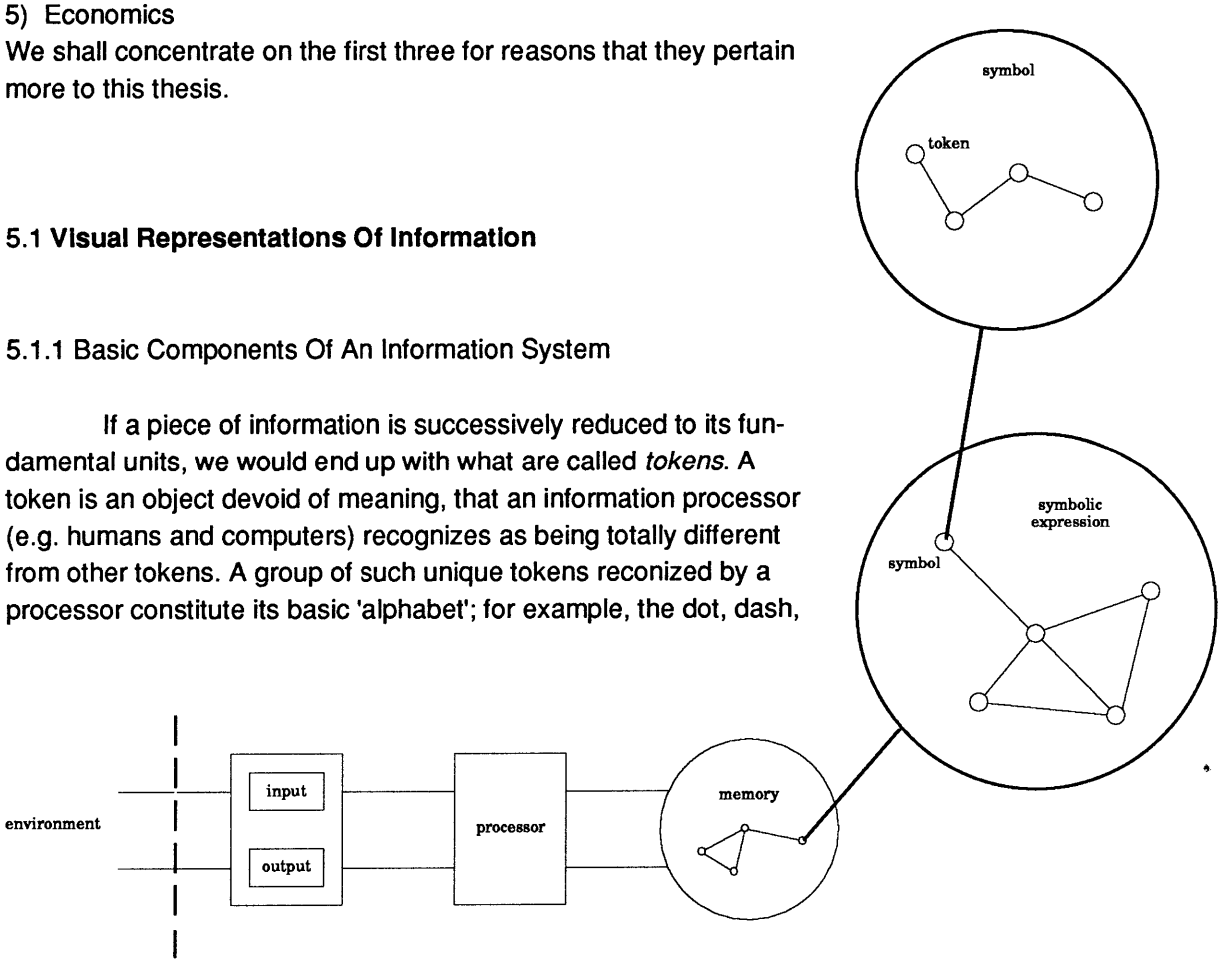
We shall concentrate on the first three for reasons that they pertain more to this thesis.

5.1 Visual Representations Of Information

5.1.1 Basic Components Of An Information System

If a piece of information is successively reduced to its fundamental units, we would end up with what are called *tokens*. A token is an object devoid of meaning, that an information processor (e.g. humans and computers) recognizes as being totally different from other tokens. A group of such unique tokens reconized by a processor constitute its basic 'alphabet'; for example, the dot, dash,

Figure 5.1 Basic structure of an information system.



and space constitute the basic token alphabet* of a Morse-code processor. Objects that carry meaning are represented by patterns of tokens called symbols. The latter combine to form symbolic expressions that constitute inputs to or outputs from information processes, and are stored in the processor memory (as data and programs).

Thus, the fundamental elements of information that we need to be concerned with are: objects (symbols, or symbolic expressions), and the relationship between these objects.

An abstract model of an information system has three features: the memory, procesor, and input-output. This abstract model of an information-processing system is representative of a broad variety of such systems, both natural and man-made. The design of information systems is primarily within the domain of engineering.

Displaying information is where the role of the visual designers is solicited - making the information accessible for human usability, that is, how humans perceive and understand information.

*The point, line, and plane, put forward by Kandinsky, could be considered the token alphabet of visual form.

5.1.2 Order

The definition of order that is relevant to a designer could be summed up by two words: commonality and compatibility. In any collection, physical objects are related by order. The ordering may be random, or by some characteristics called a *key*. Such characteristics may be intrinsic properties of the objects (e.g., size, weight, shape, or color), or they may be assigned from some agreed-upon set, such as object-class, date of entry, alignment, or orientation. Compatibility is used to describe objects that may be different, but "work" well together.

In most cases, order is imposed on a set of information for two reasons: to create their inventory, and to facilitate locating specific objects in the set. There also exist other secondary objectives for selecting a particular ordering, for example, conservation of space, and reducing vast amounts of information into maneageable quantities through such 'filters' as indexing.

5.1.3 Visibility

Visual representation of information is indispensible for two reasons. First, the limitations of human thought processes, in terms of memory and visualization capacity, requires assistance. Second,

the overwhelming channel of human perceptual input is visual.

Simply by making facts and processes, or conceptual models visible, solves several cognitive problems simultaneously:
1) The physical structure of visual representations makes the features of a problem explicit, which othewise may need exhaustive description. Visuals preserve the essential features, hence, acting as mnemonic to our thought process and easily emphasize the important features. It is easier to recognize rather than to recall from memory, an object, due to the fact that visuals act as a prompt for recognition. The displayed relationships of the features also make for easy comparison.
2) The spatially displayed features of a visual representation can show how one element, or event is accessible from, or leads to another; that is, a user can tell what is going on, and can figure out what actions are possible at any moment.
3) Simply making processes visible, the user can get instant feedback (results) of his actions.

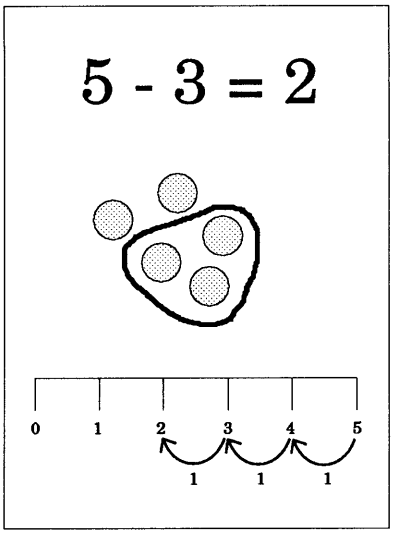


Figure 5.2 These examples demonstrate the difference between an algorithmic and a graphical technique in illustrating an abstract concept.

5.1.4 Forms of Graphical Representations

5.1.4.1 Isomorphic/Analogical representation

Isomorphic or analogical representations illustrate facts, like a cross-section of a tree trunk, or a road map. The term "isomorphic", or "analogical" representation is used to denote representations for which there is a direct structural and mesurable relation to some of the properties of the domain being represented.

Solving a problem using an isomorphic representation is often similar to performing a physical experiment on a "real-world" situation, as oppposed to obtaining the solution by an algorithmic technique applied to a symbolic description. A physical experiment, unlike a symbolic solution, can proceed without complete specification, or understanding of the problem domain. Thus, at least in part, the power of an isomorphic reprsentation resides in the fact that there is no need to make explicit the problem domain constraints and relationships, since they are captured by the structure of the representation. Even if understood, attempting to make such knowledge explicit is often impractical because of the enormous amount of detail needed to capture the many aspects of the natural world.

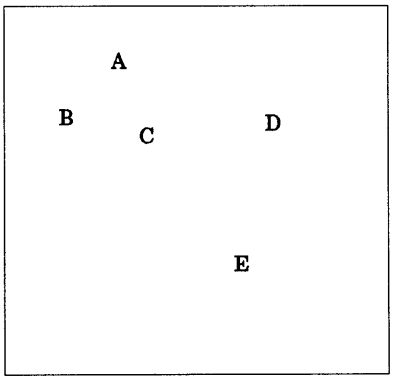


Figure 5.3 The isomorphic/analogical representation of five objects in an environment.

5.1.4.2 Relational Matrix/Feature Space

A matrix, or feature space is formed by assigning a problem related measurement to each axis of a multi-dimensional space, although it gets very difficult to visualize more than three spatial dimensions. This representation can be used for many purposes, but is especially relevant for decision making and classification tasks.

The major advantage of the matrix is the ability to handle unanticipated data relationships without pointers. Matrices are two dimensional tables consisting of rows and columns. A matrix is also conveniently represented computationally as an aray, which is a basic datatype in most programming languages. Each object, or event can have access any other object, or event in the problem space, in a direct manner without the danger of clutter. Another advantage of the matrix is that almost any kind of relationship of a comparative nature could be expressed by finding an appropriate 'weighting' for the relationship concerned.

The disadvantage of a matrix, however, is that its form often does not bear any relationship to the 'natural' form of the information space (see Mapping 5.2).

	A	B	C	D	E
A		1	1	0	0
B	1		1	0	0
C	1	1		2	3
D	0	0	2		3
E	0	0	3	3	

Figure 5.4 A morphological chart in which the quantified relationship of two object is stored in the intersection 'slot' of the two axes. If the value of the relationship represents the distance between two objects, the shortest distance between the starting point (A), and the goal (E) is not directly evident.

5.1.4.3 Relational-net

The morphological chart can easily be transformed into a relational net by representing the values in the matrix on the links in the relational net. The relational net is visually closer to the 'natural' problem space than the matrix, incorporating spatial dispostion (orientation and sequence) in its represention of information. This translates to better isomorphic depiction and better clarity of information.

The sequence is a simple linear relational net (the orderly occurance of events, or placement of objects). The nodes can be of any form (shape, size, etc.), even drawings. The relationships between the nodes, expressed as links, can be graphically meaningful, for example, they could portray the mode of getting from one node to another. Refer to Appendix 5.2.1, *Routing Behavior*. The main drawback of the relational net representation is that it can very quickly become overcrowded in a large problem set (spagetti links), but dynamics (selective emphasis), as mentioned earlier, can help somewhat.

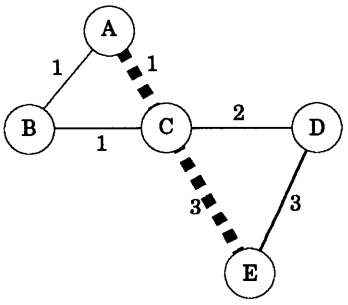


Figure 5.5 A relational net representing the same data as the morphological chart above. The highlighted links indicate explicitly the shortest route to the goal. By applying color-coding, or visual texture, the an additional dimension could be represented, for example, the mode of travel.

5.1.4.4 Tree-diagram

Relational networks, in turn, could easily be transformed into tree diagrams. A tree diagram is often used to describe exhuastively all the consequences that can arise from some initial situation, thus, requiring an exhaustive listing of all alternatives. Therefore, it is not suitable for dealing with potentially infinite problems.

Frame representation is often considered as a separate method of representation. Visually, however, it belongs to the tree representation. In a frame representation, the knowledge about the objects are stored as entries in the "slots" of the frame. The slots and frames are the nodes of the tree, equivalent to a hierarchical classification structure. Frames represent the physical or abstract attributes of objects, and in a sense define the objects. In 'scripts' events and actions, rather than objects are defined in terms of their attributes.

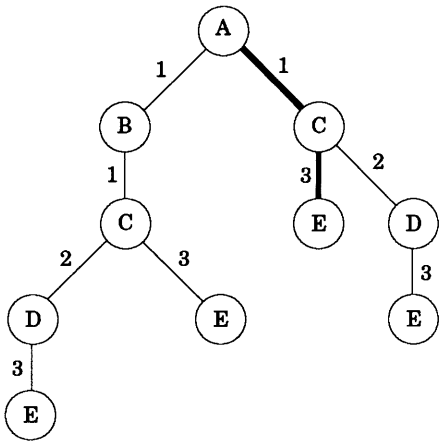


Figure 5.6 A tree diagram representing the same data as the morphological chart, and the relational net. Highlighted lines indicate the shortest path to the goal. As with the relational net, with color-coding, or visual texture to the links can provide an additional informational dimension.

5.2 Cognitive Mapping

Mapping deals with the compatibility between:
- intentions and possible actions;
- actions and their effects on the system;
- the actual system state and what is perceived by sight, sound, or feel;
- the perceived system state and the needs, intentions, and expectations of the user.

Natural mapping is the correlation between the user's mental model of the system and the system's actual operations. They should be direct as possible, with an analogical relationship between them.

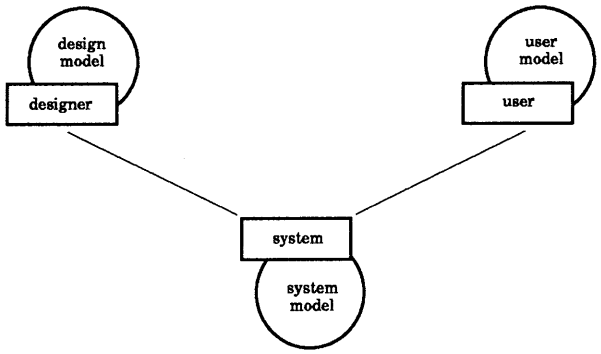
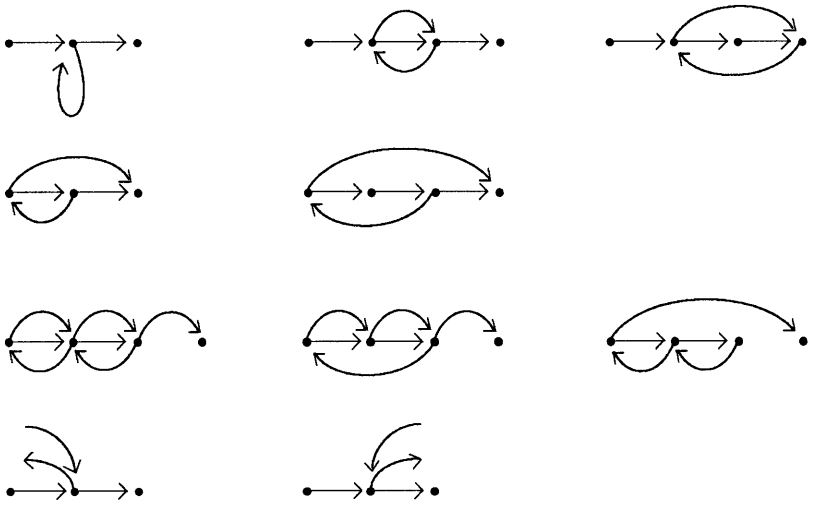


Figure 5.7 Conceptual model. The design model is the designer's conceptual model. The user's model is the mental model developed through interaction with the system. The system image results from the physical structure that has been built (including documentation, instructions, and labels). The designer expects the user's model to be identical to the design model. But the designer doesn't talk directly with the user - all communication takes place through the system image. If the system image does not make the design model clear and consistent, then the user will end up with the wrong mental model. A good conceptual model allows us to predict the effects of our actions. When a wrong model is used, we have trouble. (From *Psychology of Everyday Things*, by Donald Norman, 1986.)

5.2.1 Routing Behavior*

Routing behavior is the activity of navigating through information. To be more precise, it is necessary to distinguish between several different types of digressions. A useful catagorization scheme for this purpose is provided by Canter, Rivers, and Storr (1985) in the context of describing the navigational behavior of computer behavior through an interactive database. They were able to express the route taken by users, abstractly in terms of a set of transition between nodes representing individual location in the database, and thence to identify a number of properties inherent in the routes taken.

* David M. Frolich, *On the organisation of form-filling behavior*. Information Design Journal, Vol. 5/1, 1986.



A ring: A diversion which returns to the node from which it started.

A loop: A diversion which does not return to the node from which it strated.

A spike: A diversion which retraces at least once the path taken on the outward journey.

A scan: A diversion which involves a visit to a loosely defined collection of nodes.

Figure 5.8 Four catagories of routing actions.

5.2.4 Shape Semantics

This principle has been called variously by different groups, with minor differences, depending on their professional affliation, as affordance, body-language, and product semantics. It is defined as the symbolic qualities (cognitive meaning, symbolic functions and cultural histories) of man-made forms in the context of their use. Shape semantics provide strong clues to the operation of things. For example, buttons are for pushing, slots are for inserting, etc.

5.2.5 Other psychological aspects (learning, and ease of use)

1) Design systems that are 'retractable,' allowing for user error, change of mind, and returning to a prior state. Retractable systems encourage users to experiment in his work. Do not take control away from the user - avoid over-automation.
2) Design systems that are easy to use by keeping the operations much the same (the same interactions should be maintained at different levels of access), and not burden the user's memory with many different operational protocols.
3) When a task is or becomes complex, change the nature of the task, by transforming deep, wide structures into narrower, shallower ones e.g. tying shoelace to velcro fasteners. This example is given by Norman.
4) The function of feedback is crucial in the design of any system. It is a response by the system to the action just taken by the user. It is a dialog between the user and the system, and a measure of the current state of the system.

5.3 Formal Issues

Technology enables us to communicate better. However, we should not be mesmerized by the power of technology, and indulge in the over embellishment of information with irrelevant graphics. 'Aesthetic' considerations must blend naturally with the message, in order to effectively translate technologically enhanced information into increased productivity.

Generally speaking, strong attributes like luminance, size, color and shape are better suited for presenting dynamic information, particularly textual information. Weak attributes or low contrasts like italicizing, different typefaces, warm-cool color differences, for example, however "tasteful", are less suitable, especially when the data is dynamic, because it does not give enough time for viewers to appreciate the subtlety. When such subtleties in information are presented, they are often missed. As for that matter, anything that does not add to the communication of a message is best avoided.

5.3.1 Luminance And Color

The reason as why certain attributes are stronger could be due to the way information is processed by the human neural system. For example, visual information received is, first, analyzed for the luminance and color distribution across the retina*. Second, it is analyzed for line elements (points, edges, etc.) in the visual cortex. Then, shape perception is achieved in the temporal cortex, and motion perception is achieved in the parietal cortex.

Hence, luminance is a powerful cue for attracting attention, and is suitably employed in the highlighting of information where attention needs to be directed. However, unlike color, luminance does not have the equivalent of hue, and thus, its breadth of utility is limited.

Color which remain as the most powerful physical attribute for informaton is embedded in tradition and symbolism. Red is the traditional Chinese color for prosperity and happiness. Ironically, it is also the color for prohibition, warning, and revolution. Contemporary painting tries to free us from such fixations by emphasizing the direct sensual perceptual impact of color upon the spectator. Color is the most important and effective cue for transmitting coded meanings, and should be reserved for such purposes.

Color is cheap, when using the computer. We literally have the luxury of millions of color. Drawing from the research on color perception, although it might be possible for the average person to differentiate between 7 to 10 million different colors and shades, not more than 10 to 15 could be successfully included in a meaningful and applicable color code. America traffic signs uses twelve colors for its color code, nine of them with specified meaning*.

With computer displays, luminance and color are related. Paper is a dull reflective surface, as opposed to a monitor which is an emittive surface. A large expanse of a light color, like white, on a monitor is too glaring to look at even for a short period of time. A small source of a light color on a dark background behaves like a point source of light, creating the 'blooming' effect, which causes a loss in sharpness, or crispness of textual elements. The easy solution to this problem is to use grey instead of black or white, that is, to reduce the contrast between the visual element and the background. This, however, results in the interface having a 'greyed-out' look. The passiveness of a dull colored environment has psychological (emotionally negative) consequences.

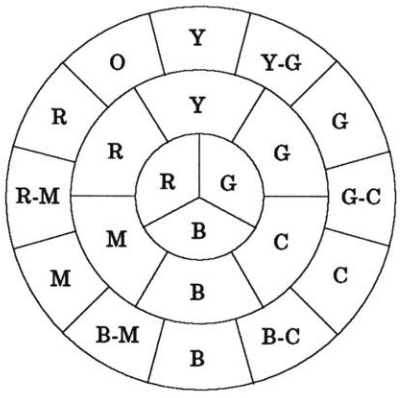
Watanabe, Yoroizawa, and Kosugi, *A Proposal of Human Interface Architecture for Advanced Information Processing Systems* (Paper in *Designing and Using Human-Computer Interfaces and Knowledge Base Systems*, edited by Salvendy and Smith).

---

*Red: stop, or prohibition,
Green: movement permitted as indicated,
Blue: general commercial services,
Yellow: warning,
Black: part-time regulation,
White: full-time regulation,
Orange: high danger,
Purple: school area,
Brown: public area,
Bright yellow-green: unassigned,
Light blue-grey: unassigned,
Coral-buff: unassigned.

5.3.1.2 Color-Coding

As mentioned above, color can carry semantic meaning. This attribute is exploited in color-coding. Color-coding is the association of a color with a object or an event. In the diagram below, starting with the primaries red, green, and blue, we can get the secondaries by mixing each pair of adjacent primaries, and the tertiary colors by mixing adjacent pairs of secondaries. With the twelve tertiary colors, the limit is reached whereby the differences between the colors are barely sufficient to make them effective as color-coders. So, not more than twelve colors (in addition to black, white, and a medium grey) could be used in an effective color-coding system. To get greater mileage from the twelve (or less) colors, they could be used in conjunction with shapes to provide distinction. Shape is described in the next section.



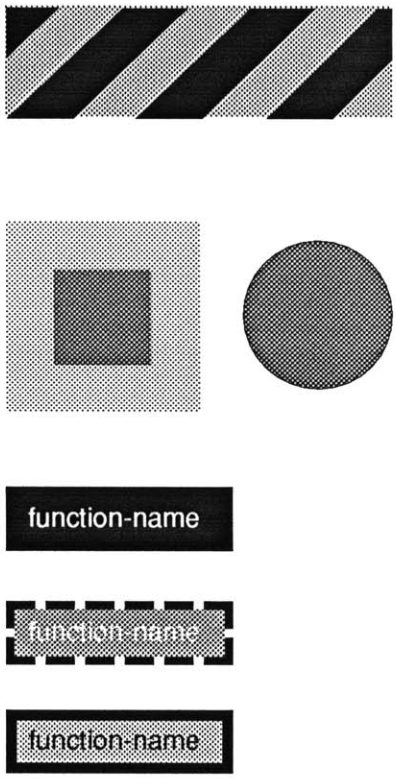


Figure 5.9 The combination of color-coding and patterns help to broaden the limitation imposed by color alone.

5.3.2 Shape

Markowitz and Dietrich* investigated the relative case of recognition of various shapes. They found that "shapes that appear to be most distinct and recognizable... are those with the most acute angles - triangles, pennants, and trapezoids." These, however, may not be the most pleasing of shapes. Birren* has shown that a consistent pairing of shape and color can be more important in the recognition of a particular sign than its legend."

The visibility of a sign is a function of its target 'value,' the combined effect of size, shape, and color. Odescalchi* found that different colored signs can have the same target value, depending on their size. The consistent use of different colors for different classes of sign is a more important factor in determining overall egibility.

* Carr, Stephen, *City Signs and Lights*. MIT Press, 1973.

5.3.3 Text

With regards to display of textual messages, the results of investigations concerned with the legibility of upper and lowercase letters are by no means conclusive. It does seem, however, that individual capital letters are more legible than lowercase letters. Hodge and Berger* have found that, regardless of various height-to-stroke width ratios, uppercase letters are legible at a greater distance than lowercase letters. Words printed in all uppercase letters were significantly more legible than those in mixed, or lowercase. However, printed passages in mixed lower and upper-case can be read considerably faster than material presented in all uppercase letters.

Forbes* and his co-workers found that lower-case had a slight advantage over uppercase when the names on the signs were familiar to the subjects. When the place names on the experimental signs were unfamiliar, uppercase gained a slight advantage. One can argue that when a subject approaches an experimental sign with a familiar name on it from beyond legibility distance, he will first perceive the shape of the word and then will be able to fill in the name without having to perceive it completely. Lowercase lettering is a help in recognizing familiar sign legend, as it provides more cues which the subject can use to infer the words. If the legend is unfamiliar, however, the subject has to read it completely - and for long enough not to make any mistakes - before being able to call out the name. Here, uppercase letters are helpful. Because of their larger size they are legible from a greater distance.

---

* Carr, Stephen, *City Signs and Lights*. MIT Press, 1973.

6.0 Bibliography

---

1. Arnheim, Rudolf, *Visual Thinking.*
University of California Press, 1969.

2. Baecker, Ronald and Marcus, Aaron, *On Enhancing the Interface to the Source Code of Computer Programs.*
Computers and Human Interaction Conference,
San Francisco, 1983.

3. Barrett, Edward, *Text, ConText, and HyperText.*
MIT Press, 1988.

4. Barrett, Edward, *The Society of Text.*
MIT Press, 1990.

5. Carr, Stephen, *City Signs and Lights.*
MIT Press, 1973.

6. diSessa, Andrea A., and Abelson, Harold, *Boxer: A Reconstructible Computational Medium.*
Communications of the ACM, Sep 86, Vol. 29, No. 9.

7. Fischer, Martin A., and Firschen, Oscar, *Intelligence: The Eye, the Brain, and the Computer.*
Addison-Wesley, 1987.

8. Gross, Mark, *Design as Exploring Constraints,*
PhD thesis, MIT,1985

9. Kalay, Yehuda E., editor, *Computability of Design.*
Wiley Interscience, 1987.

10. Kepes, Gyorgy, *The Language of Vision.*
Theobald, 1969.

11. Lakin, Fred, *Spatial Parsing For Visual Languages.*

12. Lieberman, Henry, *Design by Example.*
Unpublished paper, 1988

13. Lieberman, Henry, *Visual Programming: A Vision for the Future.*

14. Lieberman, Henry, *A Three Dimensional Representation for Program Execution.*
IEEE Conference, 1989.

15. Lieberman, Henry, *Layout as Interface: From Newspaper Design to Program Design.*
Unpublished paper.

16. Marvin Minsky, *The Society of Mind.*
Simon and Schuster, 1985.

17. Muller-Brockmann, Josef, *The Graphic Designer and His Design Problems.* Verlag A. Niggli, 1983.

18. Muller-Brockmann, Josef, *Grid Systems.*
Verlag A. Niggli, 1981.

19. Negroponte, Nicholas, *The Architecture Machine.*
MIT Press, 1970.

20. Negroponte, Nicholas, *Soft Architecture Machines.*
MIT Press, 1975.

21. Norman, Donald A., *The Psychology of Everyday Things.*
Basic Books, 1988.

22. Rand, Paul, *A Designer's Art.*
Yale University Press, 1987.

23. Rowe, Peter G., *Design Thinking.*
MIT Press, 1987.

24. Rubenstein, Richard, *Digital Typography: An Introduction to Type and Composition for Computer System Design.*
Addison-Wesley, 1988.

25. Shneiderman, B., *Direct Manipulation: A Step Beyond Programming Languages.*
IEEE Computer; Vol. 19, No. 8 (Aug. 1983), pp. 57-69.

26. Shu, Nan C., *Visual Programming.*
Van Nostrand Reinhold, 1988.

27. Simon, Herbert A., *The Science of the Artificial.*
2nd edition, MIT Press, 1981.

28. Winston, Henry P., *Artificial Intelligence.*
2nd edition. Addison-Wesley, 1984.

29. Wirth, Kurt, *Drawing, A Creative Process.*
ABC Verlag, Zurich, 1976.