UNIVERSITY OF CALIFORNIA, SAN DIEGO

Lessons Learned from Applying HCI Techniques to the Redesign of a User Interface

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science
in Computer Science

by

Jenny Lynne Cabaniss

Committee in charge:

Professor William G. Griswold, Chairperson
Professor Paul R. Kube
Professor Ramamohan Paturi

1997

The thesis of Jenny Lynne Cabaniss approved:

_____

_____

_____

Chair

University of California, San Diego

1997

To my parents

TABLE OF CONTENTS

LIST OF FIGURES

Acknowledgements

I would like to thank my advisor, Bill Griswold, for teaching and guiding me. His experience and insights had a great impact on the re-design of the interface. I also want to thank him for his patience and encouragement.

I would like to thank Morrison Chen for creating the C Star Diagram restructuring planning tool. The amount of work involved in creating the original tool is amazing. Re-designing the interface was an iterative process. I would like to thank all the user participants for providing useful feedback and suggestions for improving interactions with the interface. I would also like to thank Van Nguyen for helping to motivate me to finish my thesis.

I would especially like to thank my parents for their support, I could not have done it without them. Their attitude towards academia was a great source of inspiration. I would also like to thank my mom for helping me with the writing process by listening to my endless hours of explanations.

ABSTRACT OF THE THESIS

Lessons Learned from Applying HCI Techniques
to the Redesign of a User Interface
by

Jenny Lynne Cabaniss
Master of Science in Computer Science
University of California, San Diego, 1997
Professor William G. Griswold, Chair

In the last decade, the power of computers has risen, while the costs have rapidly declined. With the abundance of software products currently being developed, users are less likely to spend time learning a complex interface. It is important that software developers not only create beneficial services, but also create usable systems. There has been much research in the area of Human Computer Interface (HCI), yet there are still programs that suffer from poor interface designs. Chen's C Star Diagram restructuring planning tool is an example of a useful program that can benefit from an improved interface design. Chen's user-studies revealed many problems with the interface design. We looked to the HCI principles of consistency, visibility, the use of language, and work-flow scenarios to explain the problems seen in this interface. In this research, we have redesigned the C Star Diagram restructuring planning tool interface with the help of HCI principles and methods. We found that, many of the issues we faced involved trade-offs between the different principles of HCI design as well as the limitations of current technology. We have created a set of domain-specific rules of design to address these problems and help us redesign the interface. Our research provides an iterative approach to the redesign of an existing interface.

Chapter I

Introduction

In the last decade, the power of computers has risen, while their costs have rapidly declined. Computers have infiltrated the work place and the average user is not as likely to understand the technical issues behind software as a computer user did a decade ago. As software is made available to a wider group of users, usability becomes extremely important. With the abundance of software products currently being developed, users have less of an understanding of the underlying structure and mechanics of a product and are less likely to spend time learning a complex interface. Software developers not only have a responsibility to create beneficial services, but to also create usable systems.

Although there has been much research in the area of Human Computer Interaction (HCI), there are still programs that suffer from poor interface designs. Many of these programs provide useful functionality, but are difficult to understand because of inconsistent and confusing interfaces.

The C Star Diagram restructuring planning tool is an example of a useful program that can benefit from an improved interface design. The C Star Diagram restructuring planning tool is used to assist programmers plan the restructuring of large C programs. The tool builds a graphical representation of a program based on elements in the program that need to be modified. This tool helps programmers see the impact of their proposed changes before any modifications

occur. It also eases the restructuring process by helping programmers break down their task into easily manageable subtasks. Although this tool provides a useful service to programmers of large systems, user studies revealed problems with the interface design. Programmers are confused by the use of esoteric language, hidden functionality and inconsistent window design.

Our goal has been to re-design the C Star Diagram restructuring planning tool interface. We hypothesized that Human Computer Interface (HCI) principles and methodologies could direct us in creating a new interface. We learned that, although these methods are helpful, many of the issues we faced involved trade-offs between the different principles of HCI design as well as fundamental characteristics of the problems and limitations of current technology. Studying the existing interface and using HCI methodologies to address these trade-offs, we were able to design an improved interface.

I.A Background on the Star Diagram

As a large legacy system is maintained, its internal structure deteriorates. This makes further maintenance and enhancements difficult and costly. It eventually becomes necessary to restructure the system. Unfortunately, restructuring a large system is a difficult and error prone activity.

Chen's C Star Diagram restructuring planning tool was created to help programmers plan the restructuring of large systems. Many times, large C programs have globally accessible data. If the structure of this data changes, it can be very time consuming to modify all references throughout the code. By restructuring the program to access this data through a module, later changes to its structure will be localized and easier to implement.

The star diagram is a tree-like graphical representation of variable's definitions and uses in the system. The programmer can plan the restructuring of a system by addressing each unique use of the variable as presented in the star

diagram. The tool allows the user to make remarks about each kind of use of the variable. This support enables the programmer to comment on all planned changes to the system. The user can create multiple star diagrams based on different variables, and can also create a single star diagram based on multiple variables. These features can help the programmer visualize the interactions between a group of variables and their uses throughout a program. The tool allows users to navigate to the source code from the star diagram to explore details as necessary. The tool is made up of 5 kinds of interacting windows: the main window, the star diagram window, the star diagram manager window, the text window, and the search windows.

With very large systems, it is impossible to see a complete star diagram on a single screen. Nguyen created an advanced elision capability that can temporarily remove irrelevant or extraneous information from the graphical view of the program [Nguyen, 1997]. This capability allows users to customize the star diagram display and disregard certain aspects of the system representations based on the user's specific task. This added complexity, in part, motivated the interface re-design.

I.B Problems with the Interface Design

Chen's C Star Diagram restructuring planning tool is a working prototype. Although there is much useful functionality in the tool, user studies [Chen, 1996] revealed many problems with the interface's design. The interface has problems with consistency, visibility, the use of esoteric language, and poor work-flow scenarios.

The C Star Diagram restructuring planning tool interface has *problems with consistency* across windows. For example, some windows provide functionality with buttons, while other windows use buttons and pull-down menus. This inconsistency causes problems when windows have a common function, but it is

placed in different areas of each window. The **Dismiss**[1] option is in the top right hand corner of all windows with buttons, but is in the top left hand corner of all windows with pull-down menus.

Many functions are *not easily visible* to users. Users can be unaware of functionality because it is hidden in pull-down menus that are not labeled suggestively. An example of this is the string search capability in the text window. For a user to search text for a certain string, they must choose the option, **Search For String** under the **misc** menu option. It is common for users to forget that string searching is considered a "miscellaneous" function.

The tool also makes use of *esoteric and inconsistent language* on buttons and pull-down menus. The initial window prompts the user to **Add AST Files.** This button mentions Abstract Syntax Trees (ASTs) because the tool reads in text files and creates an AST representation of the program. This button is prompting the user to add all files to the tool that make up the user program. Users express worry that they are adding incorrect files.

Users have difficulty with some tasks because they take multiple repetitive steps, or the steps are not obvious to the user. These are *problems with work-flow scenarios.* An example of this is the elision capabilities in the star diagram window. The user must perform multiple steps that involve selecting a node twice to remove it from a star diagram.

There are many times that these problems overlap. An example of conflicts between visibility and work flow is the inherent problem of window proliferation. As a user explores the system, many windows are created to examine star diagrams and program text. The user wants to see all the windows they have displayed, but there is only so much screen space. Our challenge has been to resolve these conflicts.

[1]Button names are in bold font to distinguish them from the regular text.

I.C Our New Design

After looking at the user study conducted by Chen [Chen, 1996], we see that the existing interface design has some shortcomings and the users can benefit from an improved interface design. When designing a user interface, one tries to follow general user interface design principles. But following these guidelines is not enough to create a good interface. The guidelines are quite general and do not say how they should fit together in an overall design. Each system has its own complex interactions and constraints that are not accounted for in general methodologies.

Our first attempts at using HCI principles for improving the interface were inadequate. Because we began with an existing interface design, we started by trying to look at the individual problems and fix them in isolation. We realized that this was not the right approach and we turned to the concept of the *user model* to help direct us in creating a more unified, comprehensible system.

By looking to the user model to help guide us, we were able to resolve conflicts between the principles and create a tool with better work flow and better user understanding of the capabilities of the tool. Our approach to redesigning the star diagram's interface was a multi-step process. We studied the existing star diagram tool's functionality and the introduction of Nguyen's elision functionality. With the help of the user model, we were able to come up with an interface that took into account many engineering trade-offs. To evaluate our new design and show improvements in user interaction, we compare user studies performed by us and Chen.

I.D Overview of the Thesis

Following this introduction, section 2 describes the principles and methods currently in place for interface design. Section 3 analyzes the problems with Chen's interface design along with the added complication of the elision features. Section 4 explains our process and solution for redesigning the interface. Section 5

presents our findings from user studies conducted on our new design. Conclusions are reported in section 6, in which we summarize our research and discuss ideas for further research in this area.

Chapter II

Discussion of HCI Methods

In this chapter we review the Human Computer Interaction (HCI) principles, theories, and methods that have helped guide us in improving the star diagram's interface. In particular, we examine the principles of consistency, visibility, use of language, and work flow scenarios. We will also look at the differences between the engineering model and the user model and how the user model concept is applicable to our interface design.

II.A Interface Design Principles

Theories are formed from a group of failures and successes. As we approached the redesign of the star diagram we looked to the principles and theories of HCI as a way of explaining how the star diagram is successful or unsuccessful. We read a considerable body of HCI literature, and found the principles of consistency, visibility, use of clear language, and work flow scenarios to be the most helpful in creating a new interface design for a tool like ours. Although these principles are extremely general, they have helped us explain many of the problems seen in Chen's interface, and directed us in creating an improved interface.

II.B Consistency

When creating a user interface, consistency is one of the most obvious yet ambiguous goals. Webster's Dictionary defines *consistency* as the "harmony of parts or features to one another or a whole." As a user works with a system, a mental model is created. When a user encounters a new interaction with the system, previously knowledge is used to guess at how this new functionality will work. If there is no consistency between activities, the user must create a more complex view of the system, or try to remember the multiple models needed to work with the system [Brown, 1988, page 9].

When trying to implement a consistent interface, it is important to make sure that consistency is correctly applied to the interface [Brown, 1988, pages 21–24]. Aspects of the system that behave similarly should look similar. By having similar aspects of the system look and react similarly, the user can make assumptions about newly encountered functionality in the system or simply improve the user's recall regarding infrequently used features. This helps to create better understanding of the system.

Consistent Window Layout. The issue of consistency amongst similar items is pertinent to the lay-out of windows in the tool. There should be conceptual integrity amongst all windows in an interface. Similar functionality amongst windows should be consistently placed. For example, if all windows have a **Dismiss,** it should be in the same place in all windows [Brown, 1988, page 32].

The interface should also have a consistent way for users to interact with all windows of the tool. If some windows use pull-down menus, then all windows should use pull-down menus. Likewise, if many windows are created with buttons, all windows should have this form of interaction. It is confusing to have some windows with very visible functionality yet other windows have their functionality hidden in pull-down menus.

Consistent Language. There should be consistent use of language between interacting windows. If the user chooses an option from one window, the following window should have a title with a similar name to the option selected. This helps reassure users that the window being displayed is correct based on their selection [Brown, 1988, page 101].

Also, all windows should use the same word choice for the same functionality. It is distracting to use different words such as **Dismiss, Exit,** and **Cancel** in different windows, when they all mean the same thing.

Although consistency is a crucial principle in any interface design, it is important to remember that creating a system where everything looks similar can lead to confusion. Functions that behave similarly should look similar, while functions that behave differently should look different [MacLennan, 1987].

II.C Visibility

It is important to give users enough information about their current task that they can make an informed decision about how to proceed. This has been called the Answer-Question Paradigm [Owen, 1986]. It addresses issues of (1) when information is presented, (2) what type of information is presented (3) how it is represented. This paradigm relies on the fact that recognition is more powerful than recall [Preece, 1994, pages 118–119]. The user should receive suggestions from the tool as to what their next activity can be. The information should not be too intrusive and interfere with a user's activities, but it should also be easily obtained if the user needs it.

Status Information. Many times, programs have status information. This information should be easily available and visible to the user, but should not be intrusive. One alternative is to display the status information on the perimeters of a window. Many times users are not aware that status information is available and will not actively search for it.

II.D Language

When deciding on word choice one should follow general principles of conciseness, clarity, and consistency [Brown, 1988, pages 21–24]. A designer should also avoid the use of technical jargon because it can confuse or intimidate users. The designer should look at the audience of a user interface to determine what types of terms will be clear and meaningful. Users brings a certain amount of knowledge to their interactions with the interface. It is important for the developer of the tool to look at the common knowledge of the user-community.

II.E User Scenarios

Scenarios are personalized, fictional stories with characters, and events [Carroll, 1991, page 81]. They are extremely useful in helping the developer anticipate how users will interact with the tool. They help designers understand the ramifications that certain design decisions will have on user interaction with the tool. It is important to examine different scenarios to get a variety of viewpoints of a system. Scenarios can bring out new requirements of the system and see possible problem spots in the interface [Preece, 1994, page 462] [Carroll, 1991, pages 293–295].

Common Scenarios. It is important to include especially common scenarios. When interface designers recognize common scenarios, they can streamline them, helping to lessen the user's frustration and mistakes and increase productivity. Unusual scenarios can have slightly more complicated interactions.

It is important, however, to not create a system that is based solely on a list of standard scenarios. Users continually surprise developers and use a tool in unanticipated ways. If a tool is only usable with a certain set of scenarios, it would be very limiting [Carroll, 1991, pages 293–295].

II.F The Engineering Model and User Model

HCI methodology recognizes two distinct models of a given system, the engineering model and the user model [Norman, 1986]. The *engineering model* represents the way in which the system engineer conceptualizes and realizes the tool. The *user model* represents the way in which a user conceptualizes the tool. When designing a system, an engineer wants easy access to the internals of the system and wants all functionality made available in the interface. Exposure to the internals of the system aid in testing but can make it difficult for the typical user to understand the system. Users can adapt to any user interface, and as engineers create a model of a system, they are adapting to their own model, lowering awareness of its inadequacies for its intended users. The engineering model can confuse a typical user, who has no knowledge of the internals of the system.

The user model characterizes how a user conceptualizes a tool. As users work with a tool, they are creating a model of the tool based on the work they are performing and on the way in which the tool presents functionality. In creating scenarios that stress user interaction with a tool, it is possible to create a much more user-centered interface design.

Although individual users create their own model of the system, it is possible to define traits of the "typical" user to help create a more user-centered design. When creating a user image, it is important to consider the audience's level of education and the types of tasks they will be performing with the tool. As part of creating a more user-centered model, functionality should be hidden if it has little meaning to the user or can be dangerous if used incorrectly.

Chapter III

Analysis of the Existing Interface

In this chapter we discuss the original Star Diagram. We describe the C Star Diagram restructuring planning tool and its original interface. We examine some of the problems seen with the interface based on personal observation and the user studies conducted by Chen [Chen, 1996]. We also look at the added elision functionality introduced by Nguyen [Nguyen, 1997] and the effect this has had on the interface design.

III.A Description of the Star Diagram

The Original Star Diagram Restructuring Tool. Bowdidge's star diagram is a graphical tool to assist in understanding and restructuring C programs [Bowdidge, 1995]. The star diagram builds a tree-like graphical representation of a data structure and its definition and uses in a program (Figure III.1). The root node of a star diagram represents a data structure to be modified. The children of the root node represent language constructs in the program that directly reference the data structure. The children of the next level represent references to the previous level of nodes, and so on. The references to the root node are expanded out to the function level. A node having a "stacked" appearance (see the **list-ref** node in figure III.1) indicates there are at least two statements that use the parent node

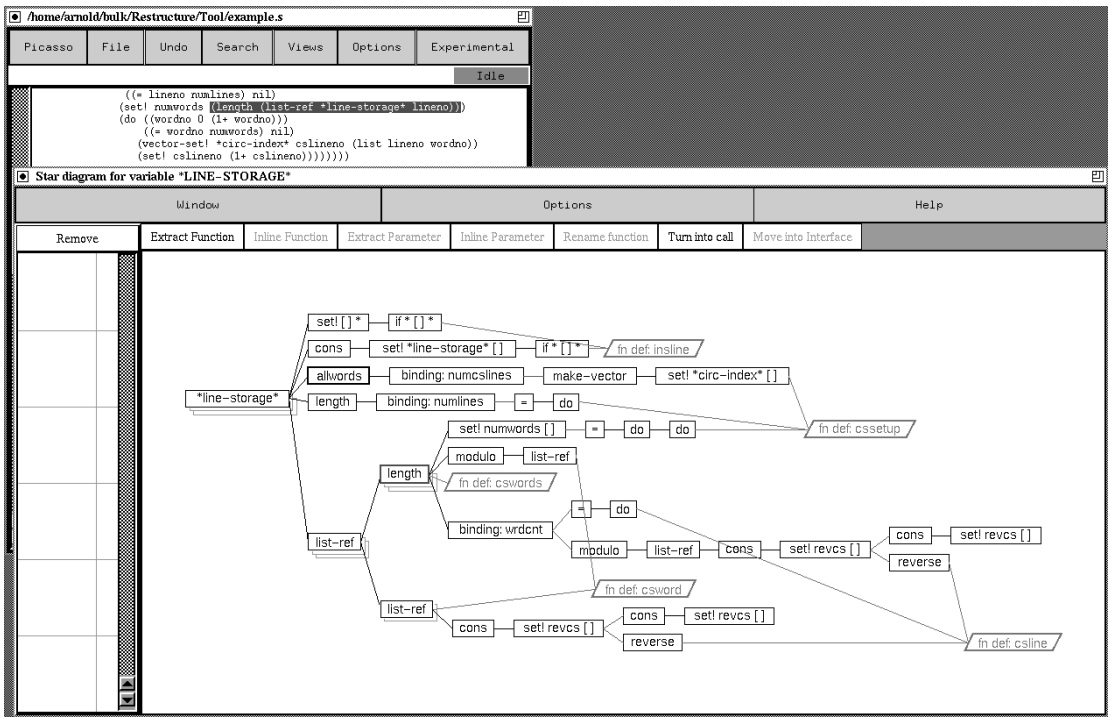

Figure III.1: An Example of the Original Star Diagram Interface.

and have the same language constructs.

To allow for a better understanding of the star diagram display, each node in the star diagram is linked to a text view. In the case of "stacked" nodes, the user can choose from a list of single-line text entries that are connected to the text views.

To perform a meaning-preserving restructuring transformation on the program, the user selects a node and applies a transformation to it. The tool checks to make sure the change will leave the program's functionality unchanged, and applies the transformation. The types of transformations a user can perform include creating a function from a group of statements, renaming an existing function, and inlining a function.

Bowdidge's tool is useful for planning and carrying out small restructuring tasks, but has problems when dealing with large systems. Examination of star

diagrams for large C programs reveals that viewing a star diagram for widely used variables in a large program is problematic [Griswold et al., 1996] because the star diagram can be quite large, compromising comprehension. Also, for very large programs, transformations can be too time consuming.

The C Star Diagram restructuring planning tool. To make the tool useful with large programs, Chen modified the star diagram by removing the meaning-preserving restructuring capabilities and turned it into a planning restructuring tool [Chen, 1996] [Griswold et al., 1996]. By replacing the tool's transformational capabilities with a plan-recording capability, Chen minimized the computation-intensive activities of the tool, allowing it to be used on larger programs with better performance.

Additionally, to help users see more of the program representation on a single screen, Chen modified the tool to allow temporary removal of nodes from the star diagram. Users can perform vertical and horizontal elisions to remove extraneous information. A vertical elision removes a node and all paths that run through it. An elided path can be displayed in isolation in a smaller star diagram window. Once the node and its paths are removed, the remaining star diagram is much smaller and the layout is more compact. Horizontal elision removes nodes from the right-hand side of the screen after a certain depth, leaving the function and file nodes. In one of Bowdidge's and Chen's user studies [Griswold et al., 1996], it was observed that users work from left to right in the star diagram, and much of the time the nodes on the right-hand side of the screen can temporarily be removed.

Nguyen has continued to enhance the C Star Diagram restructuring planning tool's capability for displaying large programs by creating a more advanced way for users to remove extraneous nodes from a star diagram [Nguyen, 1997]. Users can now elide a node based on the kind of node it is or a user specified string. For example, the user can trim all nodes that are "case" statements or the

user can trim all nodes that have a user specified string, such as "LHS." The tool continues to support horizontal and vertical elisions.

The C Star Diagram restructuring planning tool is a prototype developed to determine if planning restructuring without automated transformations is helpful to programmers. The tool is useful, but it has been observed that the tool has user interaction problems. Chen's interface is an example of an *engineering model.* The tool provides useful functionality, but is difficult to understand for users not familiar with the interface. The interface is implemented in Tcl/Tk [Ousterhout, 1994]. One of the driving factors in creating this interface was the ease of implementation based on the intrinsic properties of Tcl/Tk. For example, there are places in the interface where design decisions were based on the way that Tk handles events such as selecting and unselecting items. Once an action has been performed on a selection, it is unselected and the user must reselect the item. When creating the prototype, it was important for the designers to have access to the internals of the program when designing and testing the system. For the system to be useful to a broad range of users, we need to create a more user-centered design of the interface.

III.B Detailed Description of the Interface

The interface primarily consists of five interacting windows. In this section we will look at a general scenario and describe a typical user's interactions with the tool.

Selecting Files. The first activity the user must perform is to add files to the tool. The main window consists of five buttons (Figure III.2). The user selects the first option, **Add AST File(s),** to add the files that make up their project. The **Add AST Files** pop-up window allows the user to make selections (Figure III.3). The user can highlight files and add them to the project, or add all files in a directory by choosing the **Add ALL Files to AST** option. As the files are read

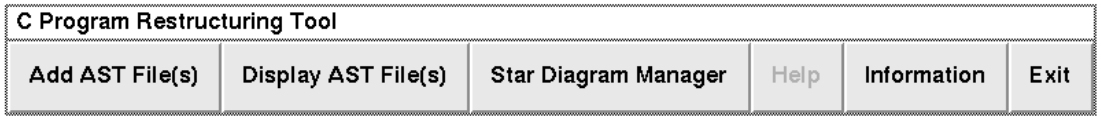

Figure III.2: Main Window.

in to the tool, they are processed. This can take a few minutes for large programs.

Looking Through Project Files. Once all files are read in, the user can begin to search through the files to find the variable(s) of interest. A file can be viewed the second option on the main window, **Display AST File(s).** This option brings up another window that will display the list of files that have been loaded into the tool and the user can select one of these files to be displayed. Each file is displayed in a text window that has pull down menus for exiting the window, creating a star diagram, and searching the file for a certain string (Figure III.5).

Selecting Star Roots. To find the variable of interest, the user looks through the files, utilizing the **Search for a string** option under the **Misc** menu on the text window. This option brings up a separate window that will search through a file (Figure III.4). In this search window, a user enters the text to search for, and either presses return or the **Highlight** button. All instances are highlighted in the corresponding text window. The user can use the **First, Next, Previous,** and **Last** buttons to navigate through the file.

Once the variable is found, the user double-clicks on the item to select it as part of the root set. The use of color helps the user distinguish between variables and non-variables. When selected, a variable is highlighted in red with blue lettering, while a non-variable is highlighted in red with black lettering. The user can select multiple star roots by continuing to double click on variables in the text view. Once a variable is added, it may not be removed from the root set.

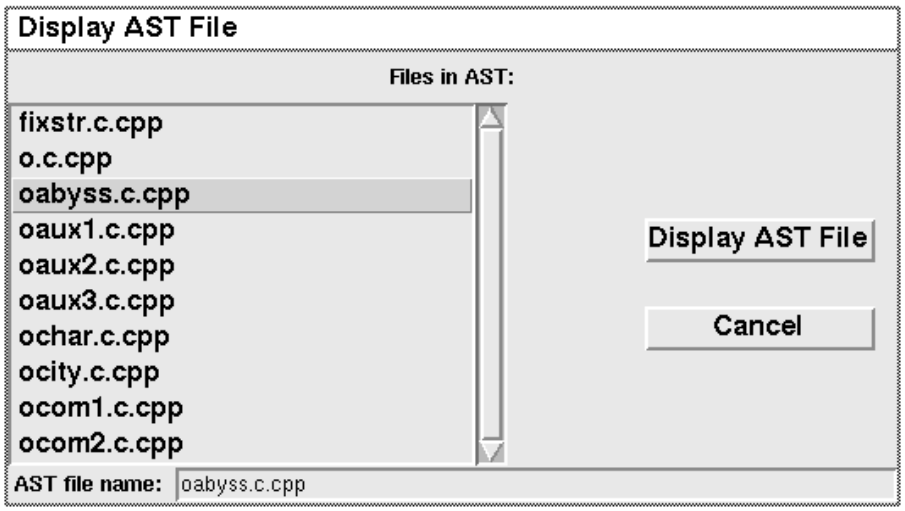

Figure III.3: Selecting a File To Display.

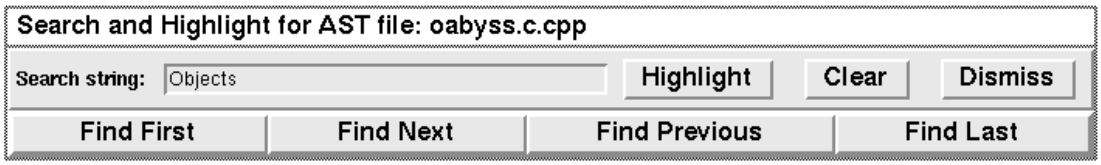

Figure III.4: The Single File Search Utility.

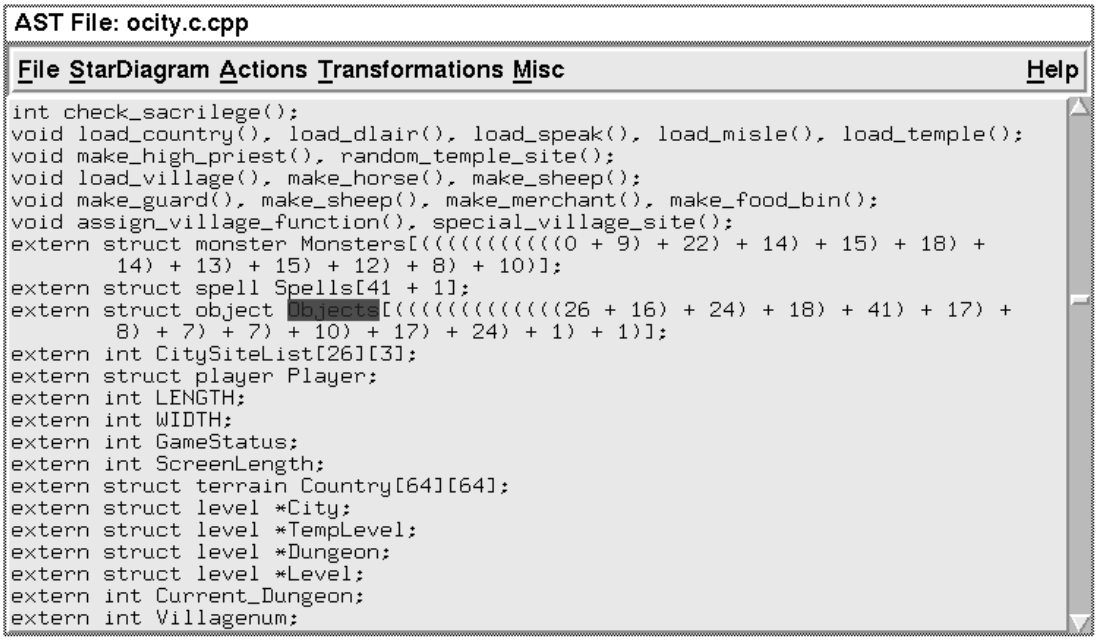

Figure III.5: The Text Window.

Creating a Star Diagram. When the user selects the **Create Star Diagram** option, the root set is displayed and the user confirms they are creating a star diagram with the listed roots. This confirmation was added because users do not realize they are adding variables to the root set by double-clicking on them. Users can end up with a star diagram that is impossible to understand because of the numerous variables in the root set. If the user has added too many roots, they may clear the complete collection of roots with the **Clear Roots** option and start the process over again.

Viewing and Manipulating a Star Diagram. The star diagram window is equipped with a side panel that will display elision information (Figure III.6). This side panel can be hidden temporarily to gain more screen space when displaying large star diagrams. Once a user creates a star diagram, they can begin planning the restructuring of their program.

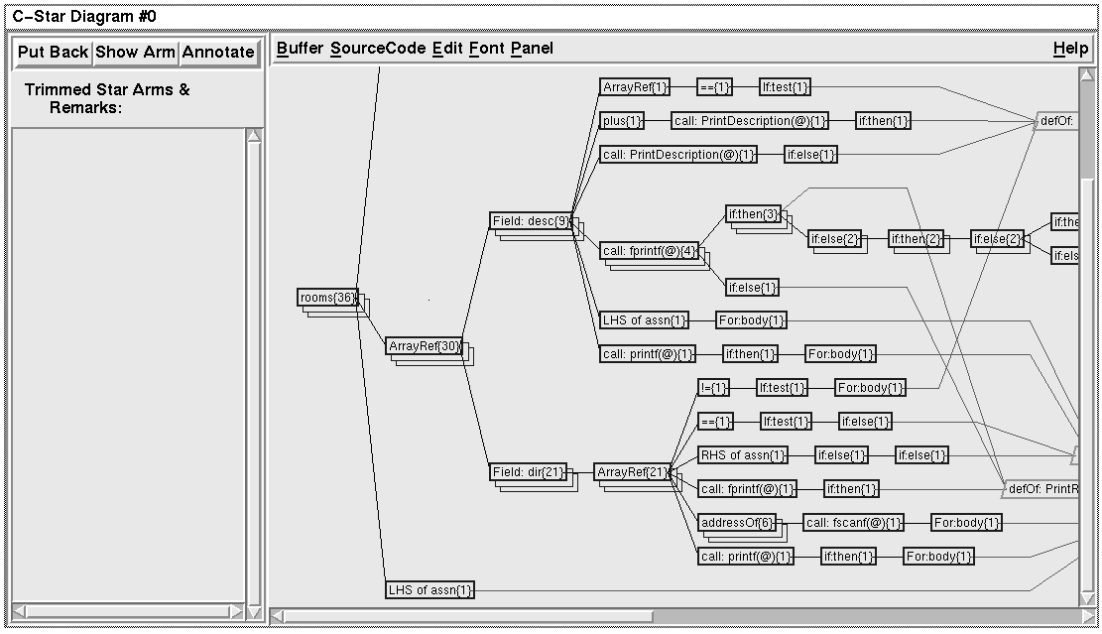

Figure III.6: The Star Diagram Window.

Status Information. When a star diagram is displayed, the user can find out how large it is by selecting the **Number of Nodes** option under the **Buffer** menu. This creates a pop-up window that displays the number of nodes in the star diagram and how many distinct paths are being displayed. This helps the user get an idea of how large a star diagram is if they can not see the whole star diagram on one screen.

Adjusting the Font for a Star Diagram. After a star diagram is created, the user can adjust the font size for the nodes in the star diagram with the **Font** option on the star diagram window. The user is allowed to enter in a font name, such as **-Adobe-Helvetica-Medium-R-Normal–*-180-*** or choose from **Small, Medium, Large** and **Huge** buttons (Figure III.7).

Corresponding Source Code. A user can look at the corresponding text for a node in the star diagram by selecting the node, and then choosing the **Show**

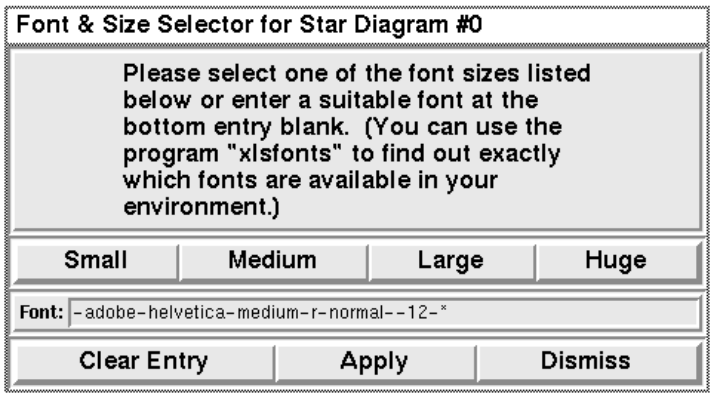

Figure III.7: Changing The Font Of A Star Diagram Display.

Corresponding Source Code option under the **Source Code** menu item. If the the node is a stacked node, a window displaying all text examples is displayed (Figure III.8). If the node is not stacked, it will bring up a text window with the corresponding text highlighted in red. A short cut for displaying the associated text is to double click on the node in the star diagram.

Annotating Star Nodes. As a user explores the star diagram a node can be annotated to help understand the functionality of that star arm in the tree. Once annotated, the information about the arm is displayed in the side panel, **Trimmed Star Arms And Remarks.** Once in this side panel, an arm can be displayed by itself in a smaller star diagram window (Figure III.9). This smaller star diagram window has a subset of the original star diagram's functionality. One cannot make further annotations or elisions on this window.

Trimming Star Arms Eliding a star arm is a multi-step process. The user must click on a node and select the **Select Star Arm** option to highlight the arm. The user then selects the node again and chooses the **Trim Selected Star Arm** option to elide the highlighted arm. Horizontal elision is similar to annotation in that a node can be selected for further description, but the arm is also removed from the

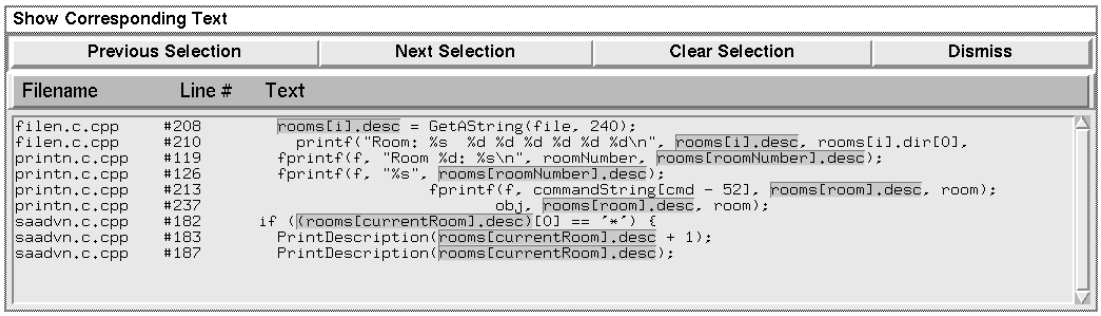

Figure III.8: AST Nodes In The Text.

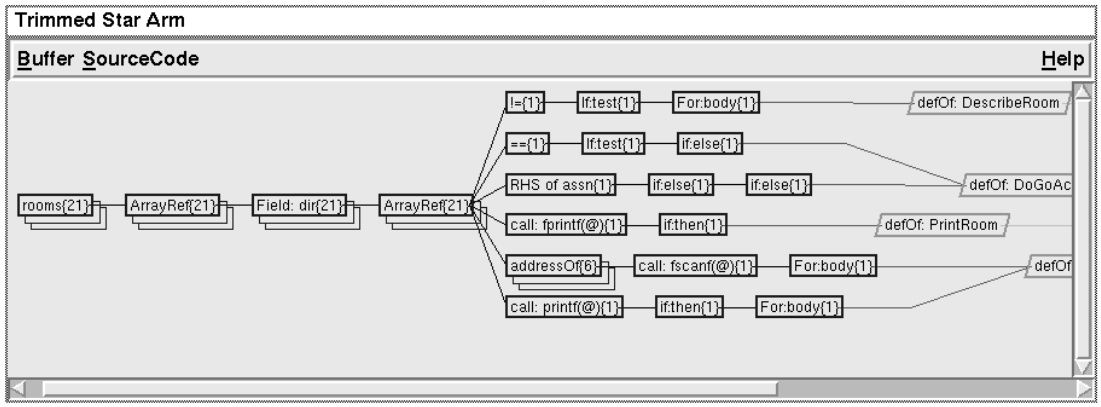

Figure III.9: The Show Arm Window.

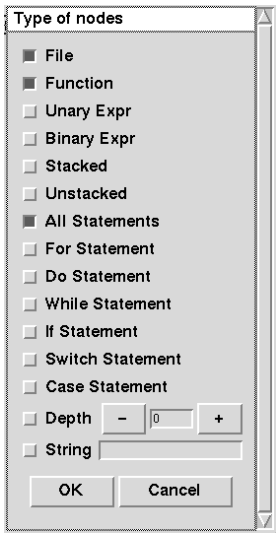

Figure III.10: The Elision Pop-Up Window.

star diagram. This helps when the user knows that a path is not relevant to the restructuring process or the path has already been considered.

Advanced Elision Capabilities. For performance reasons and the addition of new elision functionality, the star diagram is now created with only the root node displayed. The user selects the **Elision** function from one of the star diagram window menus and gets a pop-up window containing the types of nodes a user can exclude (Figure III.10). The user can then look at the elision capabilities and specify which nodes should not be displayed before the star diagram is drawn the for the first time.

Creating Multiple Star Diagrams. To better understand the impact of change on the program, a user may create multiple star diagrams. To look at previous star diagrams a user refers to the **Star Diagram Manager** window (Figure III.11). This window lists a description of all star diagrams created. Each star diagram entry has the list of root nodes used to create the star diagram and an annotation field for describing what the star diagram is used for. The star diagram's annota-

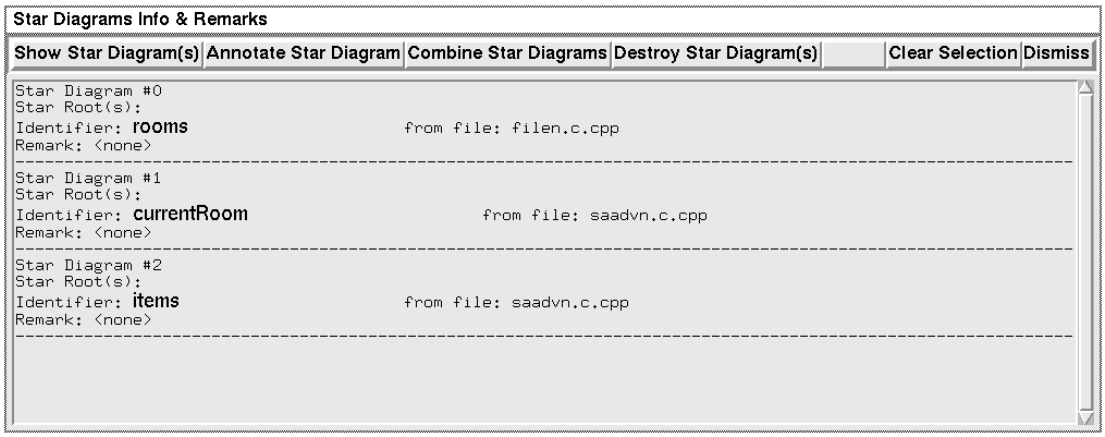

Figure III.11: The Star Diagram Manager Window.

tion field can be modified from the manager window. Each time a star diagram is created, it is added to this master list.

Display Star Diagram(s). After a star diagram has been created and dismissed from the display, it can be displayed again at a later time. Since the manager window keeps track of all star diagrams created, the user can select the star diagram and display it again from this window. This also allows users to have multiple star diagrams being displayed at any given time.

Combining Star Diagrams. From the star diagram manager window, a user can join two star diagrams to create a new star diagram whose root set is the union of the two star diagrams. This is useful when the user needs to take a look at interactions between two sets of variables.

Destroying Star Diagrams. If a star diagram is no longer relevant to the user's task, the star diagram can be destroyed permanently from the star diagram manager window. The user is asked for confirmation that the star diagram is being destroyed and there is no longer a record for the star diagram.

Chen's tool provides rich functionality, but was not designed based on the

HCI principles we described in chapter 2. By explaining the problems observed with the interface in terms of HCI principles, we hope to improve the interface design.

III.C Problems with the Interface Design

Chen's user studies of the C Star Diagram restructuring planning tool revealed many problems with the interface [Chen, 1996]. Three groups of participants were used to see the variant interactions users have with the tool. Each experiment had a group of two users working together to encapsulate the **Objects** variable in the C program, **Omega.** All groups were given a brief tutorial of the tool to help them start the task. These studies have been very helpful in revealing problems with the interface design. In the following sections we will focus on the common problems users experienced with this tool.

III.C.1 Problems Getting Started

Visibility and Use of Language. The users had difficulty with the initial stages of using the tool. The main window consists of five buttons that are difficult to understand for initial tool users. This confusion can be attributed to visibility and language problems. First time users were puzzled by the first button on the main window, labeled **Add AST File(s).** All subjects were unclear on the term **AST File.** Chen used the term **AST Files** because once the files are read in to the tool, they are used to form an abstract syntax tree (AST) representation of the program. This use of language is esoteric. It is too closely related to the engineering model, revealing the underlying implementations to the user. The real meaning this button needs to convey is for the user to add all files to the restructuring planning tool that make up the user's program.

III.C.2 Problems Finding the Objects Variable

Work Flow. Once files have been selected, the users can begin looking through the files using the text windows. Their task was to look for the **Objects** variable. In order to have a star diagram based on this variable, they had to find the variable in one of the text windows and select the variable. The groups began looking through the text windows for the variable, utilizing the **Search for a string** function under the **Misc** menu option in the text window. Since there was no **Search All Files** option in the tool it was difficult to search the whole program for the variable. In the end, all three groups resorted to the UNIX utility, **grep,** to see which files the variable was in. Since all groups initially had difficulties finding the **Objects** variable, it suggests that this type of work flow scenario had not been anticipated. The tool needs to be modified to handle this scenario successfully.

Visibility. When looking at the text windows for a search capability, one group first went to the **File** menu to find a grep-like utility. The group then searched through the menu options until they found the **Search For String** function under the right most menu option, **Misc.** The placement of this option is not consistent with other software products and caused some degree of confusion in all groups.

Use of Language. Two of the three groups had difficulties using the **Search For String** window. The options for searching the text are **Highlight** and **Clear.** Once lines containing the search string are displayed, a user can use buttons for looking at the **First, Next, Previous,** and **Last** entries. When using this window, two of the groups seemed confused about how to initiate the search. The tool searches for the string if the user hits return in the text entry portion of the window, or selects the **Highlight** button. All groups were unclear on what the **Highlight** button did. The **Highlight** button is an example of poor word choice and would be more clear if the button was labeled as **OK** or **Search** instead of **Highlight.**

III.C.3 Problems Creating a Star Diagram

Color. Once the **Objects** variable was found, the subjects needed to create a star diagram. To select the variable as a root, they double-clicked on the element. All groups questioned the use of color in the text window. After clicking on a few variables and non-variables, all groups realized what the use of color means in the text window. When a variable is selected, the lettering remains blue after another selection is made. Groups found it difficult to distinguish between selected variables, which have dark blue lettering, and unselected variables, which have black lettering, based on these subtle color differences.

The use of color also caused other problems. When the subjects tried to select root variables, they were confused by the text highlighting from a **Search for a string** command. The search for string window highlights all occurrences of a string in the text window. They assumed that since a variable was highlighted, it was selected as a root node. When they tried to create a star diagram based on this selection, the tool returned an error saying they must select an identifier to generate a star diagram. After this error message the subjects realized that they must double-click a variable to select it.

Work Flow. One group crashed the tool because they added bad roots to the star diagram. Other groups realized they had unexpectedly added roots before they displayed the star diagram. It was common practice for groups to indiscriminately double-click on variables in the text window, not realizing that by doing so they were adding these elements to the root set for a new star diagram.

Language. The font window was not heavily used by Chen's subjects, but other users have noticed problems with this window. Although this window gives the user great flexibility in creating a star diagram with any font. It is an example of where the engineering model is showing through the design. By allowing the user to enter in a font name, the tool is allowing for the maximum flexibility, but this

functionality is not very meaningful when the font names are esoteric and too long to remember for the typical user.

III.C.4 Problems with the Trimming Capabilities

Users seem pleased with the layout of the information in the main portion of the star diagram window. Much research has already been devoted to this window [Chen, 1996] [Bowdidge, 1995]. But there are still some problems with the operations on the star diagram.

Use of Language. In Chen's tool, there is esoteric language in the star diagram window. When a star diagram is displayed, the user can remove certain paths or arms of the star diagram that are of no interest or have already been processed by the user. The menu option that is used to remove these paths says, **elide star arm.** To elide something is to omit it from consideration, which is exactly what is happening in the tool, but it was observed that many tool users do not know what the word 'elide' means.

Work Flow. Users were frustrated by the multi-step process of eliding a star arm. All groups had problems with this task and a member of one group questioned aloud why they had to take so many steps to trim an arm. This was a common scenario seen with all groups, which indicates we need to modify the tool to provide easier user interaction for this process.

Visibility. Two groups initially hid the side panel to look at large star diagrams. Once they had removed the side panel, they were not sure how to get it back. Because the functions are hidden in menus, the **Show Panel** option was not easily seen by the users. After looking through most of the menu options on the star diagram window they were able to find the **Show Panel** option. This option needs to be more visible.

While trying to look for uses of the **Objects** variables, one group performed simple elisions, such as eliding all definition paths from the star diagram. Another group wanted very specific functionality from the tool. The group wanted the capability to look at just the uses of the variable on the right-hand side of an equation, but no functionality was provided for this type of elision. Nguyen added elision capabilities to handle situations like this one.

Visibility and Work Flow. The addition of Nguyen's advanced elision capabilities has caused confusion for users. With these additions, the work flow of the star diagram window has changed. When first time users look at the star diagram they are puzzled by the fact that their star diagram has only one node (the root node) in it. We observed a group of reactions to this new feature and no first time user could figure out how to get the rest of the star diagram displayed. When implementing this feature, it was easier to add the functionality to the bottom of an existing menu than create a new menu. This is an example of ease of implementation taking precedence over user understandability.

III.C.5 General Problems with the Tool

All groups were frustrated when functions took longer than a few seconds. One group wanted a stop watch displayed and all groups wanted some way of telling them that the system was still working on their last request. An example of this is when users first loaded their files into the tool. As the tool reads in a file, it processes it and adds it to the tool's internal representation. This operation can take a few minutes for large systems. All groups wondered if they were using the tool incorrectly because of the delay time. They felt much better when they saw the window that displays debug messages, showing them that the files were being processed. The subjects said they did not mind waiting, but it was important to tell them that this pause in activity is expected by the tool and is not a problem.

There was also a general feeling of confusion by one group that remained

with them for the entire experiment. After working with the tool for an hour, one member of the group commented that they still had very little understanding of how the tool works. After an hour of experimenting with a system, users should feel more confident about their understanding of the tool.

After looking at these user studies it is obvious that there are problems with user understanding of the tool. Now that we understand why some of the tool's functionality is difficult to use, we must look at how we can fix these problems using HCI methodologies.

Chapter IV

Redesign of the Interface Based on HCI methods

In this chapter we will discuss our redesign of the interface based on Human Computer Interaction (HCI) principles and methods. We discuss how our initial approach, based on HCI principles, was not adequate for redesigning the interface. We look at the principles used in our design and what techniques we applied to our design, as well as describe our new interface.

IV.A Our Initial Approach

Our initial approach involved looking at isolated problems explained by the HCI principles of *consistency, visibility, language,* and *scenarios.* Four examples follow.

Creating Consistency. One of our initial changes was to have consistent placement of the dismiss button in all windows. In the original interface, some windows have a pull-down menu labeled **File** in the top left corner that has a **Dismiss** option, while other windows use a **Dismiss** button placed in the top right corner of the windows. To create *consistency,* one of our new design decisions requires

all windows to have a **Dismiss** button in the top right hand corner of the main windows in the interface.

Using Clear and Concise Language. Two examples of esoteric language were fixed in our first stages of redesign. We immediately fixed the right most button on the main window of the tool by removing the **AST** acronym, replacing the text **Add AST Files** with **Add Files.** There was also a problem with user understanding of the word 'elision' in the tool. In the star diagram window we replaced the text **Elide Star Arm** with **Trim Star Arm.**

Scenarios We knew that the process of trimming star arms is a time consuming activity. We tried to streamline this process. We removed the need for first selecting a star arm before annotating or trimming it. In our interface, the user double-clicks on a node, selecting and highlighting the arm, and then chooses the trim or annotate buttons.

Although these modifications are useful for our end product, we came to realize it was a shallow approach to coping with the problems of the star diagram interface. We did not have an overall theme to our redesign. Many times we would make changes and later justify them with HCI principles. These principles were applied in isolation. This initial phase had no clear unifying approach and we were not making any real progress on the redesign of the interface.

IV.B The User Model and Engineering Model

A turning point in our design process was the realization that we were working with an existing engineering model when in fact what we wanted was a user model (see section II.F). This realization led us to question all design decisions in the original interface and create a more unified model that is closer to the user model.

In most of our solutions, one can see aspects of *consistency, use of clear*

language and *visibility* in harmony. Unfortunately, there were also times these HCI principles conflicted with each other and caused complex design trade-offs. One of the most pressing trade-offs occurred between issues of screen real estate and visibility. When looking for a solution to these problems, we consulted the user model and looked at work flow scenarios to determine the best way to handle these issues.

Although the user model is only a concept, we were able to operationalize it by creating tool-specific rules of design. These rules help unify the different HCI principles and resolve conflicts to create a more user-centered interface design.

IV.C Tool-Specific Design Techniques

The new design is based on a combination of HCI principles and carefully managed design trade-offs. These design rules reflect our best understanding of the user model.

IV.C.1 Button-Based Windows

Because it is difficult to remember where functionality is located in the original tool, we replaced all menu options with buttons for a completely button-based windowing system. Also, to enforce consistent placement of similar functionality and streamline *scenarios,* we determined that all main windows will have a **Dismiss** button placed in the top right corner. As we began applying this *consistency* and *visibility* technique to our design, we discovered that there was too much functionality in some windows' menus for a simple expansion to buttons.

IV.C.2 Consolidation of Functionality

In the original design, similar functionality for an interface item can be spread out across multiple windows making it difficult for users to find and understand certain functionality. An example of this is the **Make Star Diagram**

function in the text window. Making a star diagram is not related to a text window, and should be grouped with other star diagram functionality. To improve *visibility,* as much as possible, we enforce the rule that the functionality only acts on the window it is placed in. This will help users find functionality more easily. This design decision has also helped reduce the number of functions in windows, enabling us to move to a strictly button-based interface. This problem made us re-evaluate the placement of all functionality.

IV.C.3 Window Layout and Tiling

A big concern when redesigning the interface was the trade-off between *visibility* in a single window, screen real estate and the proliferation of windows, which can cause one window to hide others. This problem was especially prominent in the star diagram window. One way of alleviating the problem of window proliferation was to create a quick and consistent way to dismiss all windows (IV.C.1). Another way we minimized the number of windows on the screen was through tiling. Although this does not reduce the demand for screen space, related sub-windows can be raised and lowered together, easing scenarios involving window management. Tiling is especially useful in the star diagram window. In the original tool, this window has a side panel for the trimmed arms along with a pop-up window for the elision capabilities. The pop-up window causes problems because it either interferes with the display of the star diagram or is hidden by other windows. If a window requires related sub-windows, we tiled the large window to include these extra windows.

When creating the layout of tiled windows, we used *scenarios* to assist us in ordering the functionality and related data based on how people process information in a left-to-right, top-down manner and on the order in which the users will need the functionality.

Even though tiling is useful, we had to be careful not to create windows with too much information in them. A user can comprehend only a certain amount

of information at one time and a window with too much functionality becomes cluttered and incomprehensible [Brown, 1988, page 37]. We did not create windows with more than four tiled sub-windows.

There has also been a *visibility* trade-off with the use of tiling. Although tiling helps a large window and its sub-windows become more visible, it interferes with the ability to see other windows. We realized the window tiling might be a problem with the star diagram window and have tried to minimize the problem by allowing the user to temporarily remove tiled portions of the window.

IV.C.4 Graying

There are many times in the original interface where users are allowed to select a function but receive an error message if they cannot perform the function. These error messages explain what the user did wrong. A user must acknowledge the error message by selecting the **OK** button before they can continue. To *avoid incorrect usage* or the tool and *increase visibility* of viable options, we gray out all buttons that can cause error pop-ups at a given stage in the user's activities.

IV.C.5 Streamlining Work Flow

The original tool has many repetitive multi-step processes. We streamlined these work flow scenarios to eliminate extraneous steps and increase user work flow.

Short cuts are important to expert tool users. We have not created a formal way of designing short cuts. They were implemented after basic interface functionality was implemented. We observed common usages of the tool and found ways to expedite the activities.

IV.D The Redesigned Interface

In the following sections we describe the windows for the redesigned interface. The redesigned tool has four main windows along with other sub-windows that will be described here.

IV.D.1 Redesigning the Main Window

We began with redesigning the main window. We used a new metaphor for this window, the project (Figure IV.1). The project is not the program being restructured, but the task of planning the restructuring of the program. This window displays all project-related information.

Project Window Layout. The project window is an example of a tiled window. We combined the pop-up windows from the original main window into a single, tiled project window. This window is made up of four distinct sections, general project information, a directory listing, a file listing, and a star diagram listing. All functionality in the Project Window is laid out in a top-to-bottom, left-to-right order based on the typical user's need for functionality. The caption is placed in the top left corner to encourage users to enter a caption for their restructuring task. The other buttons along the top of the window are the **Load, Save, Help,** and **Exit.** Although help is not implemented in this version of the tool, we provided the button so that help functionality can easily be added to the interface at a later time. Users must select a directory before they can select files. After the directories are added, files must be added. And after directories and files have been added to the project, the user can begin creating star diagrams and will need the star diagram manager options, which are placed at the bottom of the project window.

Project Window Graying. The project window utilizes graying. When the project window is first displayed, the only available options are **Set Caption,**

Figure IV.1: The Project Window.

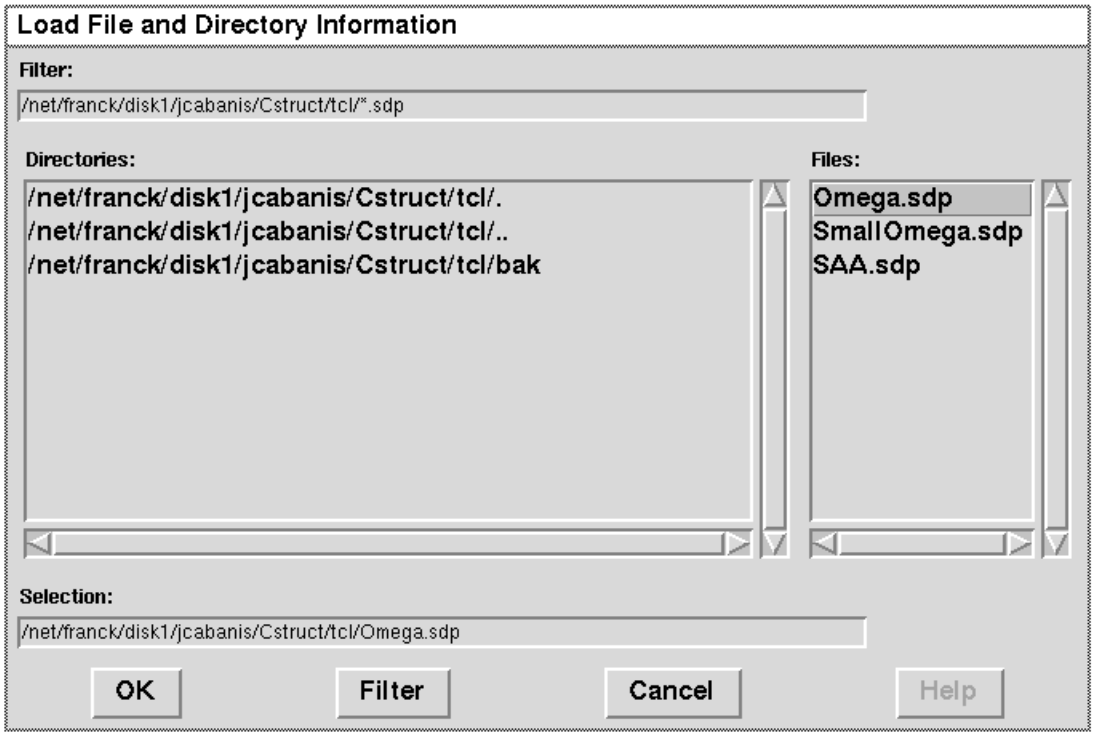

Figure IV.2: Loading A File Into The Project Window.

Load, Save, Exit, and **Add Directories.** As the user enters more information about a project, more options become available. Once there is at least one directory in the directory box, the **Add Files** option becomes available to the user. After files are added to the project, the **Display File** and **Search For A String In All Files** buttons become available to the user. After a star diagram is created, the star diagram manager functions are ungrayed.

Caption. The caption can be entered by either selecting the button **Set Caption** or double-clicking on the caption entry box. The original interface design does not have a caption field. This was a simple addition to help the user define their task. This can also help a user understand a previous restructuring project when using the **Load** option.

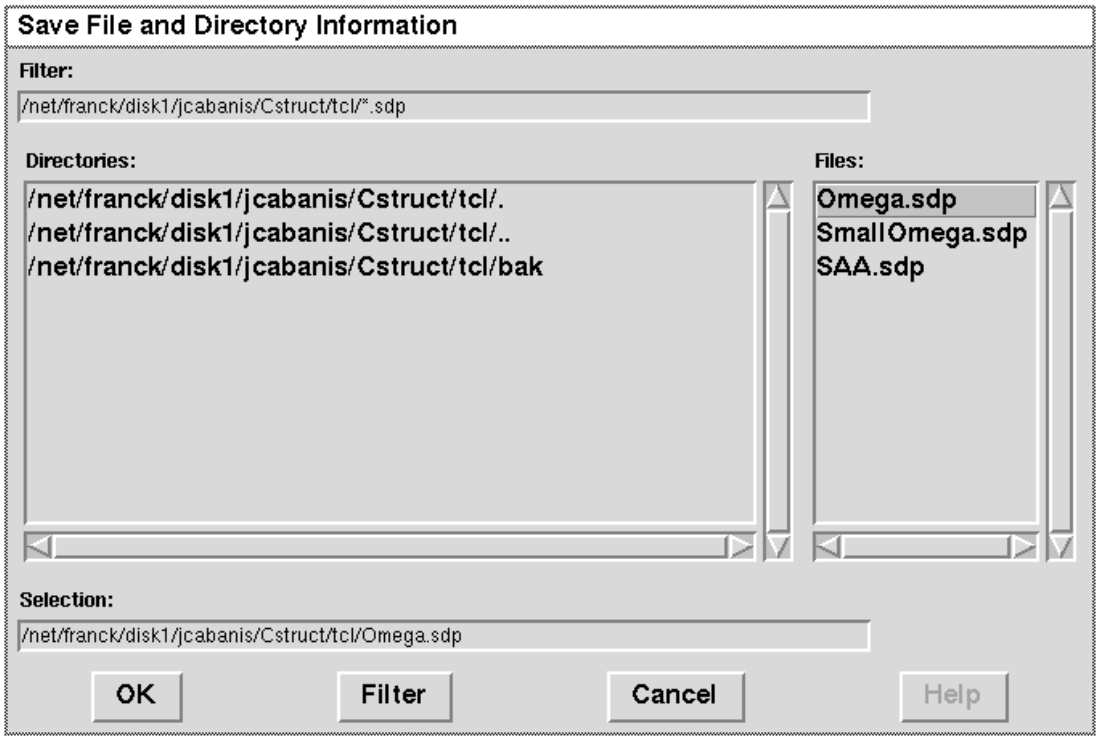

Figure IV.3: Saving Project Information.

Load a Project. The user can load the caption, directory and file information with the **Load** option from the project window. A pop-up window is brought up that gives the user a choice as to what file they load from (Figure IV.2). To help users identify previous planning files we have the naming convention that all project files ending with the **.sdp** (Star Diagram Planning) suffix are automatically recognized by the tool.

Save a Project. The user can save caption, directory and file information with the **Save** option from the main window. A window is brought up that gives the user a choice as to what file they save to (Figure IV.3).

Add Directories. Many times with large programs, the source code is divided between multiple directories. The directory portion of the project window allows

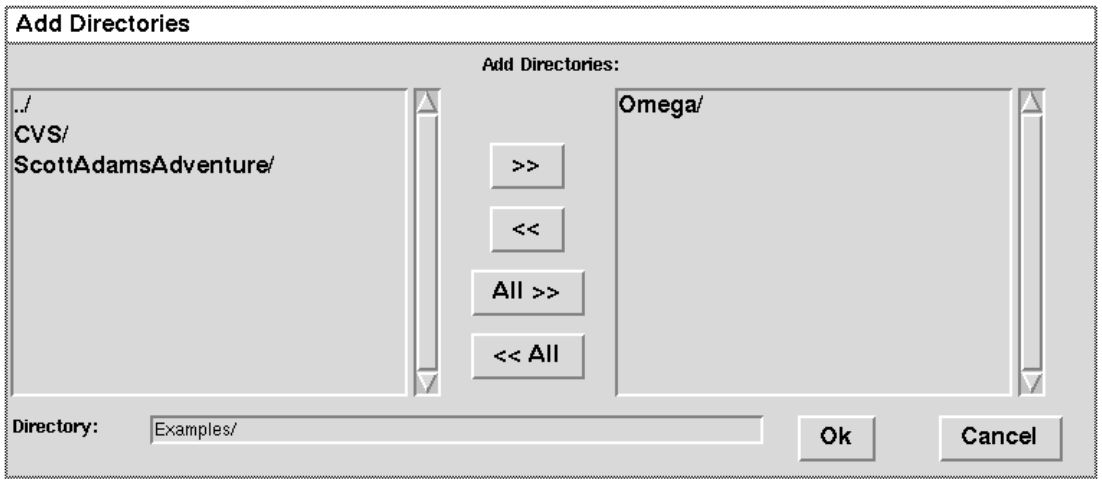

Figure IV.4: Adding The Omega Directory To A Project.

the user to add a collection of directories. A pop-up window is used to select the directories. The user can browse through the directory hierarchy and select and deselect directories with the >> and << buttons, respectively (Figure IV.4).

Add Files. After selecting the directories needed, the user selects the files to be used in restructuring planning. When the user selects the **Add Files** option, a pop-up window is displayed, similar to the directory selection process (Figure IV.5). If there are multiple directories, the user must first select a directory and then add files from that directory. The process must be repeated for all project directories.

Display File. Once files are added to the tool, the user can display the files. Only the files from a single directory are displayed in the project window at any given time. The user selects a directory and the file box changes to display the files from the selected directory. To look at a file, the user selects the file and the **Display File** button (Figure IV.6), or simply double-click on the desired file entry as a short cut.

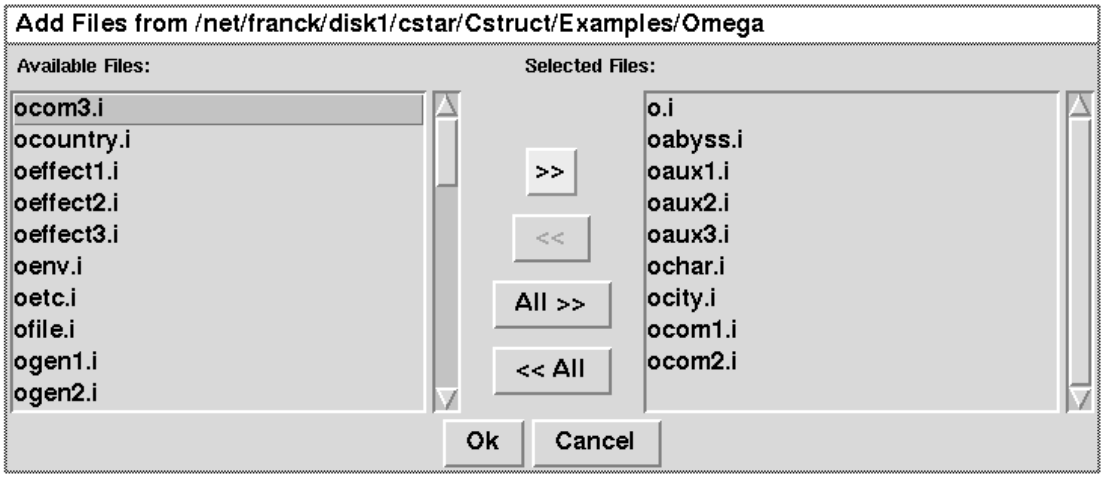

Figure IV.5: Adding Files To A Project Based On The Omega Directory.

Search for a String in all Files. If a user is not sure where a variable is located they can search for the variable in all files using the **Search For A String In All Files** button. This is new functionality. We had difficulty deciding where to place this functionality. If the search function was placed in the text windows, users would assume that it acts on a single text window. But placing the search function above the file list in the project window also has its problems. The **Display Files** button in the project window acts on a single file and it is easy for users to assume that if the search function is also placed above the list of files, it too should act on a single file. We placed the function on the project window for *visibility* and *consistency* reasons, and since the function is slightly out of place in this window, we clarified the meaning of the function by labeling it with slightly more verbose *language*. The **Search For A String In All Files** button creates a pop-up window that prompts the user for a string (Figure IV.8). The string does not need to be a complete variable name, it can be any sequence of characters.

Star Diagram Manager Functionality. For *visibility* reasons, we moved the star diagram manager functionality to the project window. The manager functionality includes displaying star diagrams, annotating a star diagram, combining star

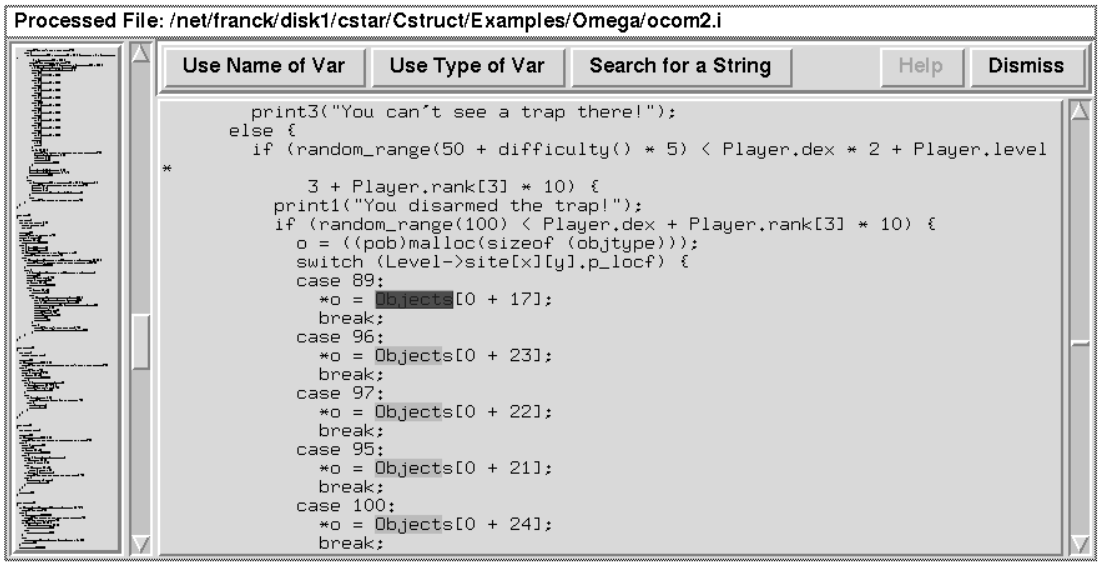

Figure IV.6: A Text Window.

diagrams, deselecting star diagrams, and destroying star diagrams.

Consolidation of Functionality. Because of our consolidation of functionality technique, we were able to combine two functions behind a single button in the project window. Both the process of creating a star diagram for the first time and displaying an existing star diagram result in the display of a star diagram. We removed the **Create Star Diagram** function from the text window and combined it with the **Display Star Diagram** function at the bottom of the project window. We modified this function to perform a simple check and if the star diagram has not already been created, it creates the star diagram and then displays it. We also added the ability to display star diagrams by double-clicking over the entry in the project window. This short cut seems to have two uses. The first use is that it helps the advanced user quickly create a star diagram. The other use was not apparent in the design phase. When the screen becomes crowded with windows, the user needs to only see the bottom of the project window in order to double-click on the highlighted star diagram selection.

We created a short cut for displaying star diagrams. Instead of selecting a star diagram and then choosing the **Display Star Diagram** button, the user can just double-click on the star diagram entry. This short cut seems to have two uses. The first use is that it helps the advanced user quickly create a star diagram. The other use was not apparent in the design phase. When the screen becomes crowded with windows, the user does not need to bring the project window to the forefront in order to find the **Display Star Diagram** function. They need to see only the bottom portion of the window to double-click on the highlighted star diagram selection.

Use of Language. We changed the text used in some of the buttons in the star diagram manager section of the project window because of spatial constraints. Many of the options contained the redundant use of the phrase 'Star Diagram,' such as **Annotate Star Diagram** and **Destroy Star Diagram(s).** When we moved these functions to the project window, space was limited by the width of the project window. Also, because these functions are logically grouped together and placed above the star diagram entries, the user understands that these functions will act on the star diagram entries and the phrase 'Star Diagram' is redundant and unnecessary.

We also modified the *language* in some buttons to clarify the meaning of the underlying functionality and minimize the screen space needed for these buttons. We changed the text for the **Clear Selection(s)** button to **Deselect.** This function removes all selection highlighting from the star diagram manager portion of the project window. This change had two positive aspects. The button takes up less screen real estate now, as well as creating better user understanding, as users were unsure of the meaning of **Clear.**

IV.D.2 Redesigning the Text Window

Our first step in redesigning the text window was to remove all pull-down menus and replace them with buttons (Figure IV.6). We were able to create enough space for buttons by moving most of the star diagram functionality to the project window and completely removing the transformational functionality that was left over from the original star diagram tool.

The text window kept some of the star diagram functionality by having the ability to select roots for a star diagram. We had difficulty deciding where this functionality should be placed and tried to resolve the problem with user scenarios. This functionality is kept in the text window because it indirectly acts on the text window by using the text selections from this window. It was also kept in the text window because the selection process has become more complex with Nguyen's addition functionality [Nguyen, 1997]. Star Diagrams can now be created based on the types of variables. We had to change the interface to allow the user to select a variable to be part of a root set based on its name or its type. We changed the interaction with this window such that a user can no longer double-click on a variable to select it as part of the root set. For a variable to be added to a root set, the user must select the variable and use one of the buttons, **Select Name Of Var** or **Select Type Of Var.** This modified interaction with the text window also has the advantage that users will not mistakenly add too many roots to a star diagram by randomly clicking in the text window. To further add to the users' understanding of the star diagram being created, the tool creates a star diagram entry and lists the roots of the star diagram before it is actually built.

Text Window Tiling. Another modification to the text window was the addition of a side panel that displays the same text in a very small font. This side panel was added because a team of users in one of our informal studies complained that they did not know enough information about where they were in the file. They also wanted to know how large the file was. To match up the two versions of

the file, all highlighting that occurs in the larger text window is replicated in this smaller side panel.

Text Window Graying. We used graying for the **Select Name Of Var** and **Select Type Of Var** buttons in the text window because of a shortcoming in the underlying functionality of the tool. The tool does not have the functionality to create a star diagram based on a combination of variable types and names. When creating a star diagram the user can create a star diagram based on the types of a group of variables or based on the names of a group of variables. In our design, when the user selects the first variable to be added to a root set by using **Select Name Of Var,** the **Use Type of Var** button is grayed out until the star diagram is displayed and a new star diagram can be created. The opposite functionality occurs when the user begins to create a type star diagram. When the underlying functionality is added to enable combination star diagrams, it will be easy to change the graying capabilities to allow selection of both options when creating a single star diagram.

IV.D.3 Redesigning the String Search Windows

We provide a single search utility (Figure IV.7) as well as and a project-wide search utility (Figure IV.8). These windows have the same look except for the window title and the colors used to highlight the search strings. Each type of search window has its own set of colors. The set of colors consists of light and dark uses of the same color. In the **Search All Files** window, all matches of the string are highlighted in light blue and the current instance of the string is highlighted in blue. In the **Search A Single File** window, all matches of the string are highlighted in light orange and the current instance of the string highlights in orange.

The design of the string search windows are based on the original tool's AST node text display (Figure IV.13). The *use of language* was the biggest

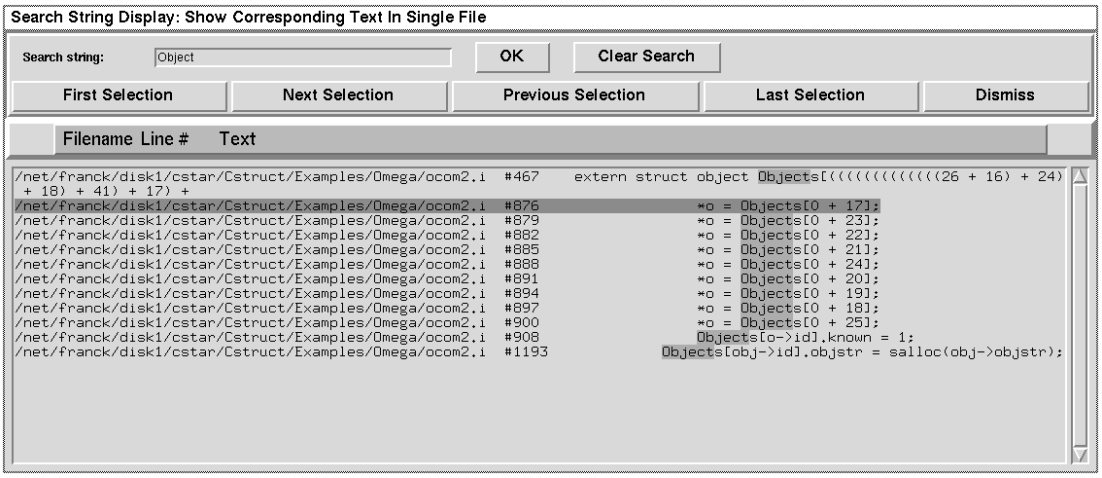

Figure IV.7: Searching For A String In A Single File.

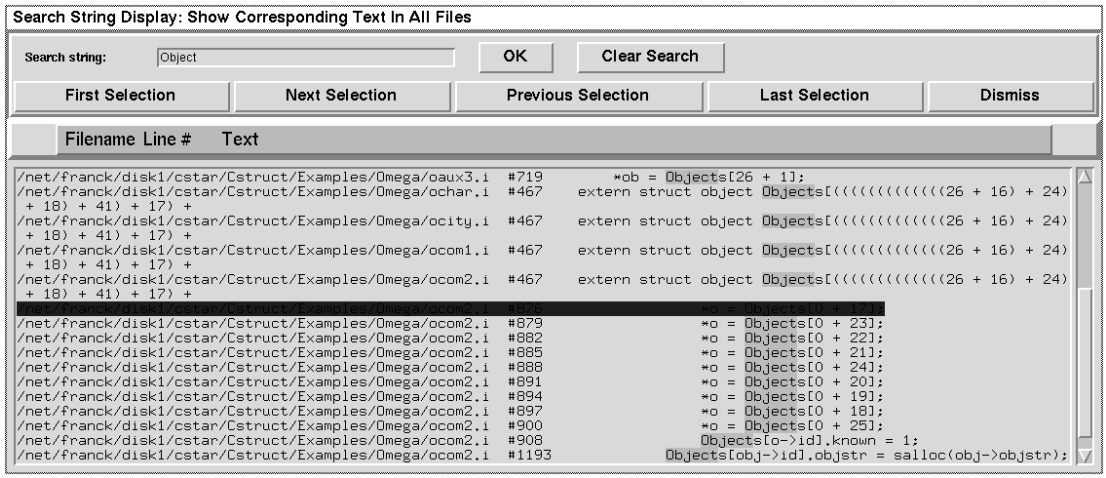

Figure IV.8: Searching For A String In All Files.

problem with the original single-file search window. Since users did not understand the meaning of the **Highlight** button in the search window, we changed the text of this button to be more consistent with other user interfaces designs and renamed it to **OK**.

IV.D.4 Redesigning the Star Diagram Window

As with other windows originally containing pull-down menus, we moved to an all-button design with the star diagram window (see Figure IV.9). We initially had too many options to place them all at the top of the screen in a button format. To create a single row of buttons, we shortened some phrases and removed extraneous steps from certain tasks. For infrequently used tasks, such as the fonts window, we kept them as pop-up windows.

Star Diagram Window Tiling. The star diagram uses tiling. With the introduction of more advanced elision capabilities [Nguyen, 1997] a floating window was initially created to handle these elision capabilities. Often, this elision window gets hidden or takes up prime screen real estate. To fix this problem we added the elision panel to the side panel of the star diagram window. We placed the elision panel just above the trimmed panel. We placed it in the top left corner to help with streamlining work flow. Typically, users work in a left-to-right, top-down, order and this placement will remind the user how to create a complete star diagram display. Both the trimmed panel and the elision panel can be temporarily hidden (Figure IV.10).

Star Window Graying. The main activities with the star diagram window that can cause incorrect use of the tool are the ability to trim arms and annotate nodes in the star diagram display. When an annotated entry in the trimmed star arm panel is selected, the path in the star diagram is highlighted. The path cannot be highlighted if it has been trimmed by another entry in the trimmed star arm panel.

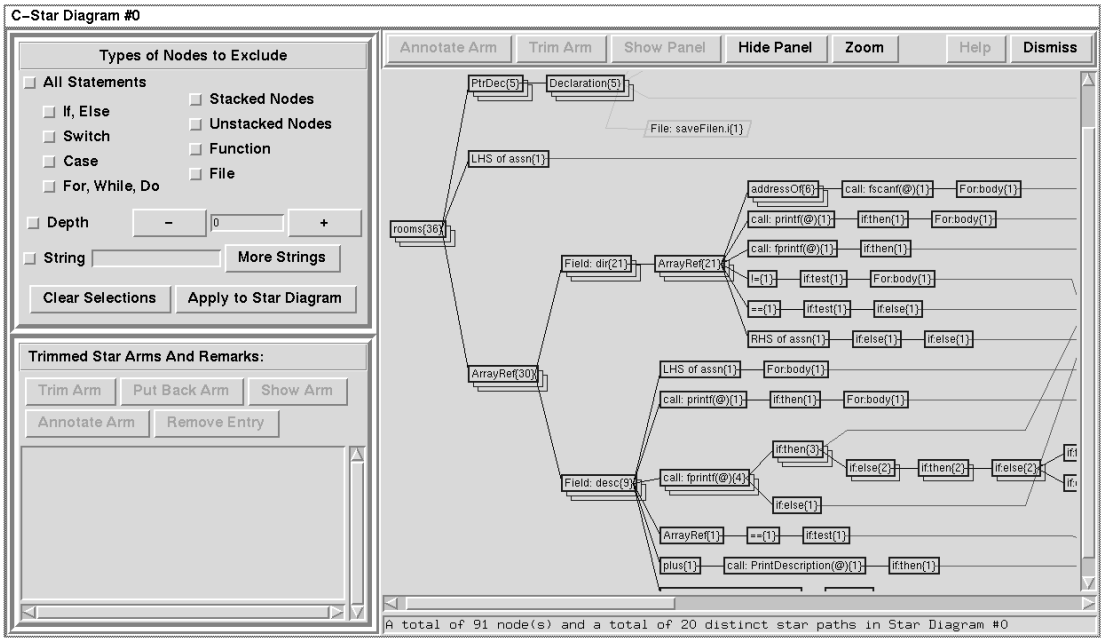

Figure IV.9: A Complete Star Diagram Window.

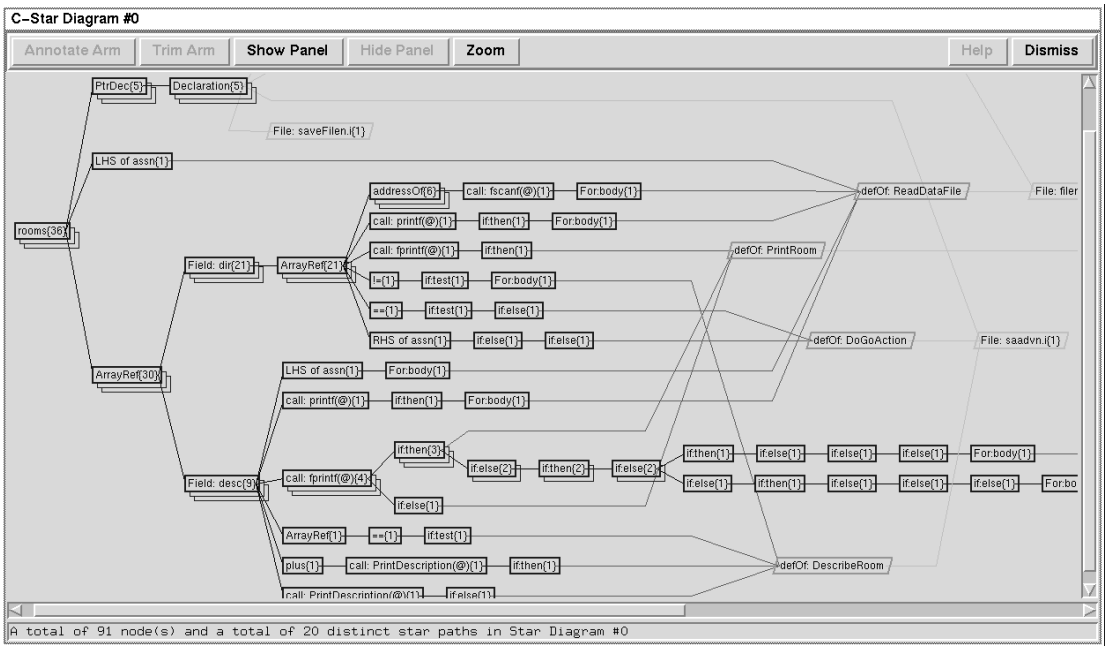

Figure IV.10: A Large Star Diagram Without the Side Panel Display.

To avoid this problem, when a user selects any node on an already annotated path, the **Trim Arm** button is grayed. Annotating and trimming can occur in both the star diagram and the trimmed panel portions of the window, and the same graying logic is placed in both sub-windows.

Streamlined Work Flow in the Star Diagram Window. We experienced some problems implementing the graying functions due to the way Tk handles selections. Once an event occurs based on a selection, the "selected" status is removed. Unfortunately, there are many times when a user wants to perform multiple functions on a selected entity, such as annotating and then trimming an arm. This is a case where the engineering model does not readily support the user model. When the tool is done with a selection, it must artificially select the element again.

Work flow scenarios have been very helpful in eliminating repetitive multi-step process from the star diagram tool. In the original interface, users are frustrated with the number of steps needed to elide a star arm. We eliminated the need to **Highlight Star Arm** before eliding the star arm: When the user selects a node, the tool automatically highlights the arm. So now the user selects the node and uses the **Trim Arm** button to elide the star arm. Shortening this work flow also helped eliminate functionality from the star diagram menus allowing easier migration to buttons.

Status Information. Instead of using a pop-up window to retrieve information about how many nodes are displayed in the star diagram and how many distinct paths are displayed, we place this information on a single line at the bottom of the star diagram window (Figure IV.9). This change was based on the *visibility* principle. The information is always displayed but does not interfere with the user's interactions with the star diagram window.

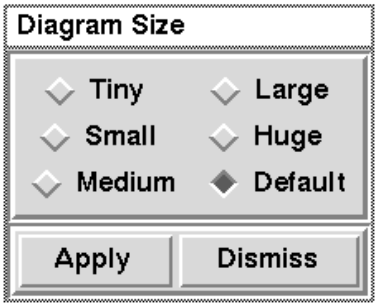

Figure IV.11: Changing The Font Size Of A Star Diagram.

Font Information. We changed the star diagram font window to have **Tiny, Small, Medium, Large, Huge** or the **Default** option(Figure IV.11). Our implementation loses flexibility because the user can no longer specify an exact font name, but this functionality is difficult to use, and is not be used by the typical user. The addition of default helps the user find the original font size if it has been forgotten or was never known.

Displaying AST Node Text. We removed the option, **Show Corresponding Source Code,** from the star diagram window and just kept the short cut of double-clicking on a node to display the source code. Many users begin by double-clicking on star nodes and never utilized the **Show Corresponding Source Code** menu option in the original tool. We also changed the window for displaying a stacked node's text (Figure IV.13) to look similar to the search windows. The main difference with this window is the change in color. To have the same look and feel as the search windows, this window also uses a two tone color approach. All matches are highlighted in pink and the current instance is highlighted in red.

Elision Panel. We also changed some of the language used in the elision panel of the star diagram window. Many users do not understand the term 'elision.' We changed the header for this sub-window to be **Types Of Nodes To Exclude.** By placing the elision panel in the top right corner of the window, users are more

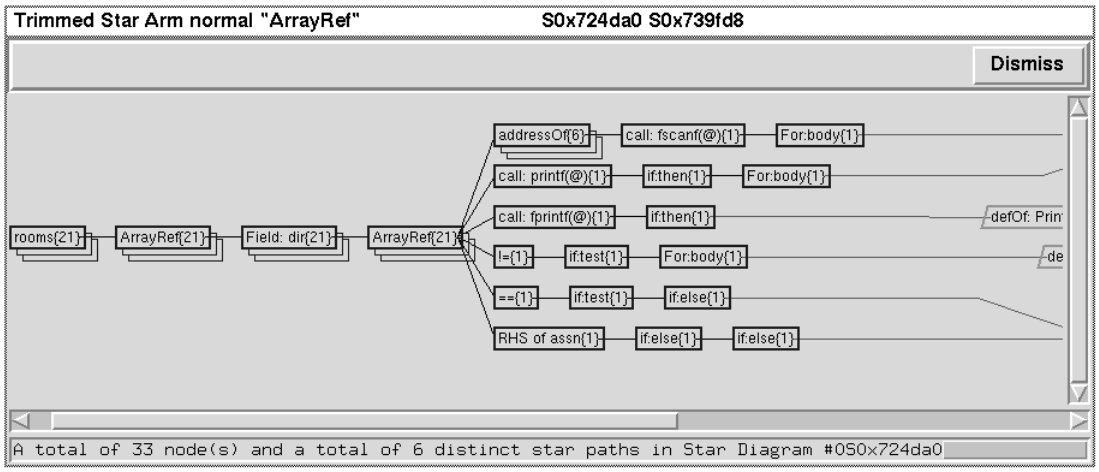

Figure IV.12: The Show Arm Window.

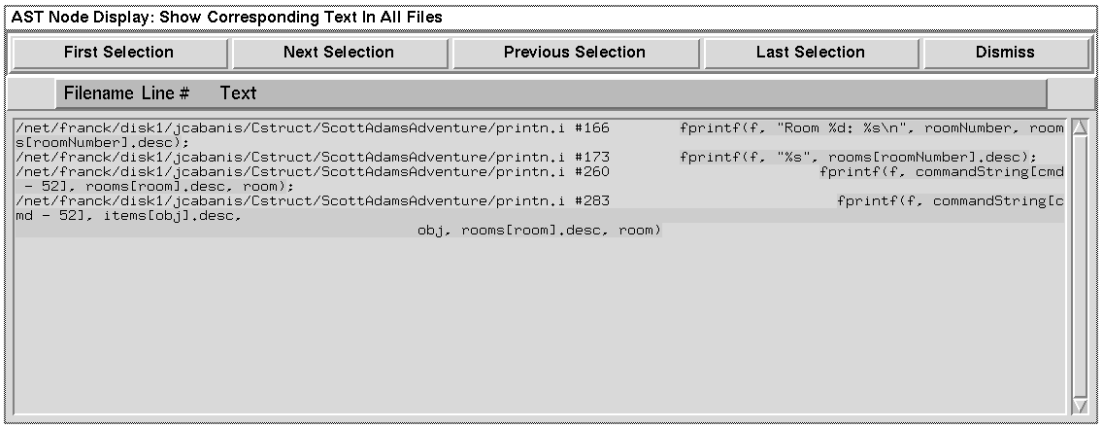

Figure IV.13: Looking At Corresponding Code Based On A Star Node.

likely to remember that they must **Clear Selections** and **Apply** in the elision panel to initially display a complete star diagram.

To improve understandability, we also removed some of the types of nodes a user can exclude. We removed the capability to trim **Unary** and **Binary** nodes because these options are not meaningful or useful in most situations.

Trimmed Star Arms and Remarks. We changed the content of the **Trimmed Star Arms And Remarks** panel to move the address of the node out of the window and display an example of the text associated with the node. The memory address of the node is meaningful to the tool developer, but is of no importance to users. We added the code example to help remind the user which node was annotated or excluded. In our design, the information kept about a trimmed or annotated star arm is the name of the node, a code example, an annotation and whether the arm is trimmed or merely annotated.

We modified the **Trimmed Star Arm And Remark** panel to fix problems with language and window size. To save space, we removed the use of the word **Star** from all buttons. To have better user understanding, we replaced **Elide** with **Trim** on the **Elide Star Arm** button.

Displaying a Star Arm. We modified the star arm display window to have the same look as the other windows by removing all pull-down menus and providing a **Dismiss** button in the top right corner (Figure IV.12). Like the larger star diagram window, we provide a status message at the bottom of the window.

Chapter V

Experiment and Results

In this section, we present a user study and an evaluation of our techniques and design for the C Star Diagram restructuring planning tool. We conducted the user study to observe user interaction with the tool. We explain the study and present the results followed by a brief analysis.

When analyzing the results of our experiment, we must be aware that there has been added functional complexity to the tool since the creation and evaluation of Chen's tool. For example, in the original interface, users can only trim star arms, whereas in our tool, the user can either trim or annotate star arms. Also, users can now build star diagrams based on the types of variables or based on the names of variables. The **Nodes To Exclude** side panel and it's functionality are also additions to our interface. These new functions have created new user interactions. Although the functionality of Chen's tool and our tool are not exactly the same, we can still gain qualitative results about changes to the interface by comparing user interactions with our interface design and Chen's interface design.

V.A Experiment

We conducted the user study to observe how users' interactions have changed with our new design. As part of our iterative development process, we conducted informal user studies prior to the formal user study.

We used Chen's user study format [Chen, 1996] for our study because it was inexpensive to use an existing experiment. Also, by using Chen's user study format, we are better able to compare our results with Chen's results. It is important to note that these results are qualitative. Our goal has been to look for changes and improvements in user interaction with our new interface design.

V.A.1 Study Subjects

One of the problems with this type of study is that it is not quantitative. It is difficult to prove the success of a new design with these types of results. We also recognize that we have few data points by only conducting one formal user study. To be consistent with the prior studies, we select a team of programmers consisting of two fellow graduate students. Both subjects were first time users of the tool. They are familiar with the concepts of object-oriented programming and modularization, but have not had a lot of experience with object-oriented programming. Although there is no average user, we believe these subjects exemplify typical behavior and are helpful in creating meaningful, though initial data points. The subjects participated in this study on a voluntary basis and neither received monetary compensation.

For the study, we had the team of programmers working together. This technique, known as constructive interaction [Miyake, 1986] [Belady and Lehman, 1976], is used in our study because it provides a natural way for programmers to discuss the problem they are solving, enabling us to observe how programmers address problems and their solutions by studying the programmers' dialogue.

V.A.2 Setup

We conducted the study in a laboratory setting to limit interruptions and ease video recording for later analysis. The two subjects worked together on a single monitor. We used a video camera to record the programmer discussion and gestures. We used keystroke capturing of computer actions for later analysis. Each subject had a clip-on microphone to get separate voice data. We recorded key strokes in all the windows used by the subjects. We video taped the computer screen to observe their activities and how their focus changes through mouse movements and selections. This also helped us see where there are pauses in the programmers' work flow. Only the subjects and experimenter were present in the laboratory during the session. The experimenter did not interact with the subjects during the video taping session, except in cases where the tool crashed or the instruction manual was incorrect.

V.A.3 Instructions

To observe the differences between Chen's star diagram interface design and our interface design, we provide the same setup and instructions as were given in Chen's user studies [Chen, 1996]. This was also done to determine how our interface design has improved user understanding of and interaction with the tool.

We first had the subjects sign a standard consent form. We then gave the subjects a copy of the instructions for the task, a ten-minute quick demonstration of the C star diagram tool and a short manual describing some of the tool's functions. We told the subjects they had 2 hours to finish their task but would be allowed to go over this time if they were close to finishing at that time. At the end of the experiment, each subject was asked to fill out a questionnaire about their background and experience in program restructuring and their knowledge of modularization. After filling out these questionnaires, they were allowed to discuss in free format their experience with the tool.

We gave the programmers an adventure program, **Omega,** written in roughly 31,000 lines of C. Omega is a rogue-like game of dungeon exploration written and freely distributed by Lawrence Brothers [Lawrence, 1989]. We asked the subjects to perform data encapsulation on the global variable **Objects.** This modification requires examining all the functions in which the **Objects** variable is used and performing several global changes. We instructed subjects to first encapsulate the data structure storing the internal representation of **Objects** (an array of **struct object**). They had to create a new module that hides the **Objects** variable behind a set of functions. We also asked that they not change any of the program's running behavior. To provide us with enough information about the tool's usage, the instructions asked the subjects to perform the restructuring as their last step.

In order to encourage use of the tool and ease analysis, there were no printed listings of the code. The subjects had to view all code on the screen. This also focused activity on the computer screen in direct view of the video camera.

V.A.4 Known Issues and Problems

In conducting informal user studies, we observed problems with user interactions with our tool. We were aware that these interactions could also cause problems for our formal user study participants.

Abundance of Color in the Text Window. Although we use color to create associations amongst windows, subjects in our informal user studies were distracted by the abundance of color in text windows that had been used by a string search window. The users assumed that if text was highlighted, it was selected. When they tried to make a star diagram based on the highlighted variables, they would receive an an error message saying that no variable was selected. We implemented the graying features to force the user to select text before they choose the **Use Name Of Var** button or the **Use Type Of Var** button. Although this has

helped, the abundance of color still makes it difficult to see which variable is actually selected.

Replicating Buttons. Because we followed the rule that functionality only acts on the window it is placed in, we had a problem with replicated functionality in the star diagram window. Both the **Star Diagram** panel and the **Trimmed Star Arms And Remarks** panel have **Trim Arm** and **Annotate Arm** functionality. When looking at user scenarios, we realized this replicated functionality might cause confusion.

Graying. Graying was one of the last items we added. We created graying to avoid adverse conditions and the proliferation of pop-up error windows. Although graying is useful in directing users to perform successfully, it also has its problems. When functionality is grayed out, the user is not really sure why they cannot perform the function, whereas with the pop-up example, the user receives an explanation of why they cannot perform the function and is usually directed to a successful scenario. Before the study we were unsure what impact this change in error handling would have on user interactions with the tool.

There were also some bugs in the graying functionality at the time the user study was conducted. By conducting the user study with this problem, we knew we would not get a completely accurate view of the subjects' perception of the graying functionality.

Status Information. Many operations can take several seconds to perform. Users of the original tool wanted status information for functions that take longer than a few seconds. Status functionality has not been fully implemented in our design and we know the users might express a need for this type of functionality.

Problems Adding Files and Directories. Many users have difficulty understanding the relationship between the directory and file sub-windows in the project

window. Users do not realize they must select a directory from the directory list before they can see the list of associated files and they become confused when all of their files are not displayed at a single time. They also have difficulty trying to add files from different directories because they have to perform an **Add Files** function for each directory. We know these interactions are a problem and we modified the experiment to have the subjects load the files from a predefined project file, **omega.sdp.**

V.B Results

In this section we describe the subjects' interactions with the tool.

V.B.1 Loading the Project File

The subjects began by loading the project file, **omega.sdp,** into the tool. The **Omega** program is composed of 47 files and takes a few minutes to load. In the tutorial, the experimenter told the subjects that this process can take some time. As they were waiting, the subjects watched the debug window to see the tool process their files.

V.B.2 Finding the Objects Variable

After loading the files into the project, they used the **Search For A String In All Files** option. The subjects searched on **Object** to look for the **Objects** variable. After receiving all uses of the **Objects variable, they began looking for the definition of the Objects** variable. In the tutorial, the experimenter did not clarify that the subjects only needed to find an instance of the variable and not the definition of the variable in order to create a star diagram.

They had problems finding the definition of **Objects** in the search window because the search window returns all instances of the variable in all files. They used the UNIX tool, **grep,** to find the definition. Once they found the definition

file, they displayed the corresponding text window and used the **Search For A String** window to locate the definition in that text window.

V.B.3 Selecting Variables and Creating a Star Diagram

The subjects had difficulty with the process of creating a star diagram. After finding the **Objects** variable in the text window, they double-clicked on the variable, highlighting it. They were not sure whether they wanted a type or name star diagram. They selected the **Use Type Of Var** button, which grayed out the **Use Name Of Var** button.

They were not sure what should happen after they selected a variable as part of the root set. The project window was covered up by the search windows and they did not see a new entry added to the star diagram manager portion of the project window. When no star diagram was created, they began dismissing search windows hoping that the project window would have some information that would help them proceed. They found the highlighted star diagram entry and selected the **Display Star Diagram** button.

After successfully creating the star diagram based on **Objects's** type, they performed a **Clear Selections** and **Apply To Star Diagram** from the elision panel. Because the star diagram was so large, it took a while to refresh the display with the full star diagram. As this was going on, the subjects realized this star diagram was not what they wanted. In the initial explanation of the tool, they were not told that they could create multiple star diagrams. So, before the type star diagram was finished displaying, they quit the tool to begin again and create a name-based star diagram.

The second time they began the tool, they understood the steps involved in creating a star diagram. As the subjects explored the text window, they double-clicked on both variables and non-variables without adding them to the root set. As part of their exploration process, they understood when and why the **Use Name Of Var** and **Use Type Of Var** buttons were grayed and activated.

V.B.4 Star Diagram Understanding

The users quickly understood the star diagram representation of their program. They began by looking at the corresponding source code for many of the nodes to match the tree-like display of their program to the actual text of their program. They also used the text editor **vi** to look at the differences between the preprocessed file in the text window and the actual **.c** file. As they were looking at the star diagram they noticed the status bar at the bottom of the window that described how many distinct nodes and paths were in the star diagram.

V.B.5 Elision Capabilities

The advanced elision capabilities were explained to the subjects so they would not be confused when they initially only saw the root node of the star diagram displayed. The subjects were also told that some nodes can be excluded immediately if they know a star diagram is going to be large. Once their star diagram was displayed, one subject recommended that because it was probably going to be large, they should display only the **File Nodes** at first. After they determined how many files referenced the **Objects** variable, they cleared all selections in the **Nodes To Exclude** panel and displayed the complete star diagram. The subjects easily understood this panel and its relationship to the star diagram portion of the window.

Later, they came back to the **Nodes To Exclude** window to experiment with the types of nodes they can exclude and how it will affect the display of the star diagram. The **Stacked Nodes** and **Unstacked Nodes** options initially caused some confusion with the subjects. All other types of nodes to exclude are based on the text inside the node, where as these options are based on the visual presentation of the node. After trying these two options, they quickly understood their meaning.

V.B.6 Restructuring Planning

Once they felt they had a good understanding of the star diagram representation, they began to discuss their plans for restructuring the program. They did not spend time trying to discover what made up the **Objects** array, and just concentrated on hiding the structure behind a set of functions. They attempted a simplistic object-oriented approach, where they would have an **Init()** function along with **Get()** and **Set()** functions[1].

They worked in a top-down fashion, looking at each arm in the star diagram. The first node they encountered was the node denoting the external definitions of the variable. To get rid of this node and its associated file nodes, they first tried to trim the file nodes connected to the external definition. They tried to remove the file nodes associated with the external definition, but were not able to. File nodes are special nodes because they have multiple paths leading into them, and hence do not uniquely identify a path to be trimmed. The subjects were finally able to remove these nodes based on the **Declaration** node connected to the file nodes.

V.B.7 Annotating and Trimming Star Arms

The subjects had difficulty with the annotation and trimming functions. This was partly due to the explanation they received in the beginning of the experiment. They were not told that once a node on a path is annotated, the path cannot be trimmed. They did not understand why they could not trim the node and tried to annotate other elements on the same path in the hopes that one of the nodes would let them trim the entire arm. The documentation was incorrect for this activity, telling the users to trim an already annotated path from the star diagram window instead of from the trimmed arm panel. The experimenter had to intervene and correct the documentation in order to allow the experiment to

[1] Although this design satisfies the minimal requirements of the instructions, it is not necessarily the best design [Griswold et al., 1996].

continue. Once the correct trim functionality was explained the subjects were able to continue with their restructuring planning.

Once they had figured out how to annotate and then trim an arm, they were able to work through the star arms very quickly. For each star arm they went through the same motions for annotating the arm. They would annotate the arm, trim the arm, and then have to adjust the star diagram display.

V.B.8 The Restructuring Process

When the subjects began the process of restructuring the code, they undertook restructuring in the same order they processed star nodes. For each entry in the trimmed panel, they displayed the arm with the **Show Arm** button. They looked at the corresponding code for the star arm and found the same code in the original text file, and began making modifications. Time ran out before they could finish the restructuring process, but they described the rest of their process to be similar to the first modification, saying they would have modified each instance in the order they annotated the paths.

V.C Interview

After the experiment, we had the subjects fill out a short questionnaire. It asked them questions about their experience with program restructuring as well as their general level of experience and understanding of concepts such as modularization. We then conducted an open-ended interview with the subjects, asking them for their reactions to the tool's interface.

V.C.1 General Comments

The interviewer began by asking the subjects their general feelings about the tool. They were then asked about problems they experience with the individual windows.

Status Information. As we guessed, the subjects did complain there was no status information for functions that take several seconds. They said they did not mind waiting for a function but wanted reassurance that the tool was still processing their request.

Graying. Because the experimenter warned them of possible problems with the graying capabilities, they did not know which graying functionality was a bug and which graying functionality was intended. Many times they assumed that the graying functionality was an error, when in fact, it was correct. They said they understood the gray-toggling between the **Trim Arm** and **Put Back Arm** functions on the **Trimmed Star Arms And Remarks** panel of the star diagram window.

When the experimenter explained that the graying was implemented to avoid pop-up error-message windows. Both subjects agreed that they prefer graying functionality over pop-up error messages in tools.

V.C.2 Project Window

For the most part, the subjects were satisfied with the project window, but they did have some problem knowing when they could use the star diagram functions. They were confused by the process of creating a star diagram and were not sure when they could begin to utilize the star diagram functions on the project window. This confusion can be partially attributed to the star diagram functionality left in the text window.

V.C.3 Search Windows

They did not feel the search windows were especially helpful but did not have any problems with them either. Because of the subjects' misunderstanding about the process of selecting the definition of a variable instead of just any instance

of a variable, they wanted the definition to be highlighted in a different color than the other entries in a search window.

V.C.4 Text Window

They liked the side panel on the text window. They also understood why only certain parts of the text are highlightable, and why the **Use Name Of Var** and **Use Type Of Var** buttons gray out.

They were disturbed by the use of red in the text window. When they looked at a star node's corresponding text, there were times when the node corresponded to a large portion of text and most of the text display was highlighted in red. They commented that, in most tools, the color red usually means danger, which is not the case in this tool.

Star Diagram Window. Both subjects said they had problems understanding the significance of color in the star diagram window. They had problems identifying which node was the selected node, even though when a node is selected, the path highlights in red and the node turns blue. One subject said the selected node should be a much brighter color, such as bright green. As another way of distinguishing the current selection, they wanted the ancestors of the selected node to be one color and its descendants to be another color. They said it was difficult to differentiate blue, selected nodes from black, unselected nodes.

The process of trimming a star arm frustrated the subjects. They had to adjust the display of the star diagram every time they trimmed a star arm because the star diagram would always be redrawn further down on the screen. In our re-design, we did not address this problem, and left the functionality unchanged from the original tool.

The subjects did not like the display for the **Trimmed Star Arms And Remarks** entries. They said it was too much information in too small of a space. They described the data as too scattered and poorly formatted. This layout was

compact because of the decrease in space the trimmed panel has due to the addition of the elision panel.

V.D Analysis of Results

In this section we discuss the experiment and its results as well as evaluating our design techniques and how they have affected user interactions with the tool.

V.D.1 Determining Successes and Failures

When analyzing qualitative results, it is difficult to clearly demonstrate successes and failures. We tried to analyze the quality of the subjects' interactions based on the amount of training they received as well as their verbal reactions to the tool and the ease in which they performed their activities.

Amount of Training. When analyzing subjects' usage of a tool, it is important to take into account the type of training they received. While we realize it is crucial to give subjects some training, it is difficult to establish how much training is necessary. Because we wanted to determine if users could easily navigate through the interface, we provided minimal training. After the study was performed, we concluded our training was inadequate for certain scenarios. There was a clear correlation between the lack of training for certain activities and the problem spots in the subjects' work flow.

Verbal Reactions. Although users find it easy to criticize aspects of an interface, they rarely comment on the ease of use of an interface. In some sense, a successful interface is "invisible" to the subject. It is still possible to gain verbal confirmation of the success of an interface by observing how the subjects integrate the language of the interface into their discussions.

Work-flow. Success can be seen in the lack of complaints from the subjects, relative to the previous studies. Also if a subject is able to quickly perform activities with little discussion as well as expeditiously recover from a confusing scenario, we can say this is a successful use of the tool. Likewise, if there is a great pause in activity or the user has difficulty finding to a successful scenario, we know there is a problem with our work-flow scenarios.

V.D.2 Project Window

On the whole, the project window concept has proven successful for our tool. There were no verbal complaints about this window, along with no confusion or great pause in activity when using the window. The window also provides benefits that were unrealized in our design phase. We observed that whenever the subjects became confused and were not sure how to proceed with their current task, they would refer to the project window in the hopes that it would help direct them. An example of this is when they initially had problems creating a star diagram and referred to the project window to find the **Display Star Diagram** button. Likewise, when the subjects were looking for a save function, they immediately went to the project window and found the **Save** button.

V.D.3 Star Diagram Window

Our modifications to the language in the star diagram window have proven useful. When the subjects discussed the **Nodes to Exclude** functions, their sentences included language from the interface, "Let's exclude the stacked nodes," showing the interface language is meaningful and useful. The users had initial problems understanding the meaning of the **Stacked Nodes** and **Unstacked Nodes** options in the **Nodes To Exclude** panel, but through exploring the functionality, were quickly able to learn what these options meant and continue the planning process.

There were problems with the tiling in the star diagram window. In our interface, the **Trimmed Star Arms And Remarks** panel is half as big as it is in the original tool. We realize this is a visibility problem within the sub-window and have tried to compact the information about each entry so more entries can be seen at one time. The users expressed confusion over the format of the entries and said it was too crowded.

Because of our rule of consolidating functionality, the subjects were confused by the duplicate functionality in the star diagram sub-windows. The subjects had difficulty making the association that both sets of annotate and trim functions act on star diagram information. This difficulty can also be attributed to incorrect training and instructions. Once the experimenter stepped in and told the subjects the correct process for annotating and trimming an arm, they were able to successfully continue.

By telling the users there were problems with some of the graying functionality, we biased their understanding of why functions are grayed in the star diagram window. When the subjects did not understand why a button was grayed out, they assumed it was one of the problems the experimenter had warned them about, when in fact, the only valid problem they experienced was a graying a problem in the **Trimmed Star Arm And Remarks** panel with the **Put Back** and **Trim Arm** functions.

V.D.4 Text Window

The side panel on the text window was well liked by the subjects and did not cause any confusion. Also, the change from pull-down menus to buttons, enabled the subjects to quickly and easily utilize the **Search For A String** button in the first text window they encountered.

The subjects had problems with the process of creating a star diagram. This partly can be attributed to the fact that there is still some star diagram functionality in the text window. The subjects became confused when they had

to move from the text window to the project window to continue the process of creating a star diagram. Most windows in the tool are associated by a "derives" relationship, where one window creates the next window the user needs. The send-back relationship between the text window and the star diagram manager portion of the project window is not as apparent. After selecting the root set from the text window, it is the user's responsibility to bring up the project window.

V.D.5 Auxiliary Windows

Our modifications to the auxiliary windows helped to create smoother work flow. The main changes in these windows involved clarifying the use of language. The subjects were easily able to navigate through the **Load, Save,** and **Annotate** windows. The subjects did not utilize the **Zoom** window, so we do not have data points on this window.

V.E Summary of Results

Although these results are based on a single pair of participants, the results proved useful. The subjects' high-level behavior, (such as the designs they chose), were similar to the high-level behavior of subjects in our informal user studies, implying that these are not unusual subjects. By analyzing the results we realized the impact our training had on the activities of the users. This study has also been helpful in confirming that our techniques were useful in creating improved interactions with the tool. We realized that, for the most part, our tiling technique improved visibility as well as eased window management. Our changes to the language of the tool have proven successful based on our observations of the subjects' conformity to the tool's language. We also noticed improvements in work flow due to the button-based window lay-out.

Our user study was also helpful in determining areas where our design techniques have not been inadequate in creating a completely successful interface.

The use of tiling improves visibility in a single window, but hinders the visibility of other, potentially useful windows, as we saw in the case of the hidden project window. Also, our rule for consolidation of functionality caused problems when a function did not have an obvious association to a single window. We have the problem of duplicate functionality in the star diagram window as well as the problem that some functionality does not easily belong to a single window, such as the **Search For A String In All Files** function in the project window and the **Use Name Of Var** and **Use Type Of Var** functions in the text window.

Chapter VI

Conclusion

With the complexity of software products on the market, users are unlikely to understand the internals of a tool. It is important for software developers to create both useful and usable systems. Chen created a planning restructuring tool to aid programmers in restructuring large legacy systems. Although the tool is useful, user studies revealed problems with user interactions.

There has been much research in the area of HCI in the last decade. User interface design is not a mechanical process, but a domain-specific process that utilizes HCI methods. We hypothesized that the HCI methods could help us in redesigning the C Star Diagram restructuring planning tool interface. We started with the basic principles of consistency, visibility, the use of clear, concise language, and work flow scenarios. Because we began with an existing interface, our initial approach did not utilize all aspects of the methodologies. After reading additional HCI literature, we were introduced to the user-model and engineering model concepts. This was a turning point in our design. We began questioning all aspects of the interface and were able to develop an improved interface design. We developed a user model in terms of a set of tool specific design rules that we applied to our tool.

VI.A Contributions of the Research

This work provides a number of specific contributions including the use of a user model, engineering trade-offs and our learning experiences.

The Engineering Model and User Model. The most important aspect in creating a new interface design was the realization of the user model. The use of the user model implies an iterative approach to interface development and was applied to our design because we began with an engineering model. Instead of improving small problems in isolation, the user model helped us to question all design decisions in the engineering model and develop a more user-centered interface design.

Trade-Offs. There were many times the HCI principles conflict with each other. We had difficult engineering trade-offs such as visibility and the proliferation of windows. These principles could not be applied one at a time, but rather all at once as seen in our design rules.

Case Study. Since a case study is not algorithmic, its relevance cannot be clearly measured. Our case study provides a useful example of the process involved in developing domain-specific design techniques. Also, our learning experiences can be useful to other interface developers by warning them of some of the problems experienced with creating and using domain-specific design techniques.

VI.B Lessons Learned

Although our techniques were useful in creating improved user interactions with the tool, there were also some unanticipated problems with our design.

Use of Graying. We introduced the concept of graying a button when underlying function could cause an error or cause problems with later tool interactions.

After removing the error message windows, we realized that both types of error handling have their merits. Error messages are able to explain why a user cannot perform a function and direct the user to a successful scenario. On the other hand, it is bothersome for users to stop their activities and dismiss the error message. Graying is good because it does not allow incorrect activity, and there is no need to dismiss extra windows. On the other hand, when buttons are grayed out, users do not always know why they cannot perform functions and receive no help in moving on to a successful scenario.

Visibility and Expansion. By creating a button-based window layout we have improved visibility, but have limited the extensibility of the tool. In a menu-based window system, it is easy to add functionality to the bottom of an existing pull-down menu. Many of the tool's windows do not have space for additional buttons. Adding functionality to the tool might involve redesigning a window.

Separation of Functionality. We tried to localize functionality. This causes problems when windows are closely inter-related. Most windows are associated by a "derives" relationship, where one window creates another as part of directing the user to the next activity in a scenario. Cases where there is a send-back relationship between windows result in user confusion. Because of separation of functionality, the text window has a send-back relationship to the project window. After a user selects variables from the text window, they then must move back to the project window to create a star diagram. First time users have problems with these interactions and we have found it difficult to make them aware of this type of relationship.

VI.C Future Work

Our research has left some unanswered questions and led us to new questions with respect to the C Star Diagram restructuring planning tool interface.

Use of Color. We added more color to the star diagram, but were not sure if the significance of the colors would be understood. Our user studies revealed that color is a distraction in many cases. Because we did not know the impact the use of color would have on user interactions, it has been modularized and can be easily changed to reflect new findings.

Status Information. Many users want the tool to display a status bar for functions that take longer than a few seconds. To implement this type of status bar requires complicated communication between the interface software, implemented in Tk, and the underlying **C++** functionality.

Short Cuts. We did not have a methodical way of implementing short cuts. We know it is important to provide short cuts for expert tool users and know there is a need for a formal approach to creating short cuts in our tool.

Interrelated Functionality. User studies reveal problems with user understanding of window relationships and interactions. The replicated annotation and trimming functionality in the star diagram window needs to be modified by either connecting the functionality behind these buttons, or only having one set of buttons. The separated star diagram functionality in the text window needs to be better connected to the rest of the star diagram functions. In creating these relations, a developer might need to modify our design techniques.

Saving Star Diagram Information. Some users have articulated scenarios that span many days due to the careful and incremental process of restructuring a large program. Currently the C Star Diagram restructuring planning tool only saves caption, directory and file information. Based on the underlying implementation, storing the star diagram representations pose many problems. Currently, the star diagram uses memory addresses as identifiers for the star nodes. There needs to be some way of removing this system dependence. Also, there are questions of

how to display a star diagram if the preprocessed files that make up a star diagram have changed. These issues need to be addressed before all tool information can be saved.

Selecting Multiple Star Nodes to Annotate or Remove. The ability to perform a function based on the selection of multiple star nodes has been requested by users in both our and Chen's user studies. To allow this type of behavior, we need to create a new design rule to enforce a standard user interaction with respect to multiple selections in all windows of the tool.

Bibliography

[Belady and Lehman, 1976] Belady, L. A. and Lehman, M. M. (1976). A model of large program development. *IBM Systems Journal*, 15(3):225–253.

[Bowdidge, 1995] Bowdidge, R. W. (1995). *Supporting the Restructuring of Data Abstractions through Manipulation of a Program Visualization.* PhD thesis, University of California, San Diego, Department of Computer Science & Engineering. Technical Report CS95-457.

[Brown, 1988] Brown, C. M. (1988). *Human-Copmuter Interface Design Guidelines.* Ablex Publishing Corporation, Norwood, New Jersey.

[Carroll, 1991] Carroll, J. (1991). *Design Interaction: Psychology at the Human-Computer Interface.* Ambridge University Press, New York, New York.

[Chen, 1996] Chen, M. I. (1996). A tool for planning the restructuring of data abstractions in large systems. Masters Thesis, University of California, San Diego, Department of Computer Science and Engineering. Technical Report CS96-472.

[Griswold et al., 1996] Griswold, W. G., Chen, M. I., Bowdidge, R. W., and Morgenthaler, J. D. (1996). Tool support for planning the restructuring of data abstractions in large systems. In *ACM SIGSOFT '96 Symposium on the Foundations of Software Engineering.*

[Lawrence, 1989] Lawrence, B. (1989). Omega [A complex, rogue-like game of dungeon exploration written and freely distributed by Lawrence Brothers.]. Copyright 1989. Available from Lawrence Brothers at **brothers@paul.rutgers.edu**.

[MacLennan, 1987] MacLennan, B. J. (1987). *Principles of Programming Languages: Design, Evaluation, and Implementation.* Holt, Rinehart, and Winston, New York, 2nd edition.

[Miyake, 1986] Miyake, N. (1986). Constructive interaction and the iterative process of understanding. *Cognitive Science*, 10(2):151–177.

[Nguyen, 1997] Nguyen, V. B. (1997). Impact of adding customizability on software architecture: A case study. Masters Thesis, University of California, San

Diego, Department of Computer Science and Engineering. Technical Report CS97-523.

[Norman, 1986] Norman, D. A. (1986). Cognitive engineering. In Norman, D. A. and Draper, S. W., editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*, chapter 3. Lawrence Erlbaum Associates, Inc.

[Ousterhout, 1994] Ousterhout, J. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA.

[Owen, 1986] Owen, D. (1986). Answers first, then questions. In Norman, D. A. and Draper, S. W., editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*, chapter 17. Lawrence Erlbaum Associates, Inc.

[Preece, 1994] Preece, J. (1994). *Human Computer Interaction*. Addison-Wesley Publishing Company, Menlo Park, California.