An Empirical Evaluation of Dependence Analysis in Parallel Program Comprehension

Douglas Monk

CRPC-TR95553-S
May 1995

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

RICE UNIVERSITY

An Empirical Evaluation of Dependence Analysis in Parallel Program Comprehension

by

Douglas M. Monk

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

Ken Kennedy, Noah Harding Professor
Computer Science
Chair

Keith Cooper
Associate Professor
Computer Science

David Lane
Associate Professor
Psychology

Scott Warren
Adjunct Assistant Professor
Computer Science

Houston, Texas

May, 1995

An Empirical Evaluation of Dependence Analysis in Parallel Program Comprehension

Douglas M. Monk

Abstract

This research contributes two advances to the field of empirical study of parallel programming: first, the introduction of the **Xbro***wser* system, a unique general-purpose hypertext/hypermedia system combining high-quality text formatting using TEX* or LATEX with author-controlled support for event-level protocol analysis and computer-assisted instruction. Second, an extensive ground-breaking empirical study using **Xbro***wser* tested effects of dependence analysis and related factors on error severity and time required for successful comprehension of loop transformations relevant to both sequential and parallel program comprehension.

The results show that graphical annotation of program source with dependence information as is done in the ParaScope parallel programming environment improves the time required to correctly comprehend the results of parallelizing loops, and that dependence type affects both time required for successful comprehension and severity of errors made in the attempt, with anti- dependences somewhat more problematic than flow dependences, and both much worse than output dependences, particularly for loop carried dependences.

An alternative to dependence analysis based on simpler data-flow concepts was not better in either comprehension time or error severity measures, nor were parallel loop transformation shown to be more difficult to understand than equivalent sequential loop transformations for this particular task.

Controlling for the previous experimental effects, GRE Analytical and Mathematics scores, SAT Mathematics scores, mathematics grade point average, number of high school and college mathematics courses, and the percentage of working time spent programming were all found to correlate to improved error and/or time performance.

---

*TEX is a trademark of the American Mathematical Society.

Acknowledgments

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

bi-smi 'l-lāhi 'r-raḥmāni 'r-raḥīm

In the name of God, the Beneficent, the Merciful

To my wife, Tazeen A. Monk, for unending support.

To Abdullah Muheisen, Fahad al-Temawy,
Mohammedi El-Hallabi, and Hasan Abdul Latif, for Islam.

To my thesis committee, for their patience:
Ken Kennedy, Keith Cooper, David Lane, and Scott Warren.

To my in-laws, for comfort and assistance beyond the call of duty:
Tosiq and Nadira Ansari,
Saad, Anas, Hasan, and Zaid Ansari,
Farheen and Zia Khan, and Ma'az,
and Noorin and Serene Ansari

To the heart of the department:
Iva Jean Jorgensen and Ellen Butler

To my brother and his family:
Richard and Joanne Monk, and Bobby and Michael.

To the memory of my father:
Richard D. Monk

To my mother, with all my love:
Oleeta Russell Monk

Finally, to my son, Muhammad Ansari Monk:
Now I'll get you those toys I've been promising you.

Contents

Illustrations

Tables

Preface

This dissertation is the culmination of several years of effort both on my part and by those on whose shoulders I stand. An interdisciplinary work integrating the fields of computer science and psychology must attempt to fulfill the desires of each; aspects of each discipline deserve more attention than is possible to include here. The primary audience of this work is assumed to be computer scientists and others interested in human factors relevant to parallel programming and dependence analysis. Every effort is made to explain terminology or to refer to explanations that the larger community interested in this research may find useful. A certain minimal amount of inevitable repetition in such explanation is made between chapters to accommodate readers who may be inclined to browse rather than assiduously scrutinize.

Chapter 1

Introduction

Dependence analysis – the study of data dependences arising from shared references to a memory location occurring in two different places in a program – is critical to understanding and constructing correct parallel programs, [KKP+81, AK87, SLY89, Hag90] yet the effects of data dependences on parallel program comprehension have not been investigated sufficiently despite rich resources in empirical techniques that have been applied to sequential programming; indeed, virtually no empirical work has been done on parallel programming [Wad93].

My research has contributed two advances to the field of empirical study of parallel programming: first, I introduced the **Xbro***wser* system, a unique hypertext/hypermedia system that combines high-quality typesetting using TEX[1] or LATEX with author-controlled support for event-level protocol analysis and computer-assisted instruction. Based on a common, freely available and popular standard for typesetting that produces display-device-independent document files, the **Xbro***wser* system uses hypertext annotations created using the flexible and extensible typesetting languages of TEX and LATEX to display text, graphics, hypertext links, and interactive editors using standard X Windows[2]. These considerable capabilities are further extended by allowing the hypertext author to control generation of an event-level protocol record of reader interaction with the hypertext documents used by the **Xbro***wser* system. These event records may then be further processed to yield data readily suitable for statistical or further protocol analysis using **Xbro***wser* or other systems; in effect this can combine the protocol transcription and segmentation phases if desired, considerably speeding protocol analysis. In addition, the hypertext scripting language used in the **Xbro***wser* system provides extensive support for computer-assisted instruction through the mechanism of session variables under author control, including flexible interactive reader text input. These three features – high quality hypertext,

[1] TEX is a trademark of the American Mathematical Society.

[2] X Windows is a trademark of M.I.T.

event-level protocol analysis, and support for computer-assisted instruction – provide an excellent platform for many uses, such as empirical user interface research, interactive automated training, and user interface prototyping.

Second, I validated the value of these features of the **Xbro***wser* system by using it in an extensive ground-breaking empirical study that tests the effects of dependence analysis and related factors on error severity and time required for successful comprehension of loop transformations relevant to both sequential and parallel program comprehension. This experiment included thorough computer-assisted training in the basics of parallel programming and dependence analysis requiring an average of 90 minutes of interactive instruction, followed by a data-collecting phase requiring about 90 minutes to complete in which twenty-six subjects worked twenty-four problems covering a variety of areas suspected of affecting performance on parallel programming. Subjects then used **Xbro***wser* to answer a roughly fifteen minute long questionnaire about many aspects of background factors possibly related to performance on parallel programming, followed by a final interactive debriefing averaging fifteen minutes explaining the purpose of the study and the correct answers to the problems, again conducted using **Xbro***wser*.

The results of this study show that an interface prototype of graphical annotation of program source with dependence information similar to what is done in the ParaScope parallel programming environment [Che93, CCH+88a, CCH+88b] improves the time required to correctly comprehend the detailed and specific results of parallelizing loops, and that specific forms of dependences have different effects on both time required for successful comprehension and the severity of errors occurring in the attempt. An alternative to traditional dependence analysis derived from simpler data-flow concepts was also tested, but was no better than the more complete traditional method, nor were parallel loop transformations more difficult to deal with than equivalent sequential loop transformations on the particular task used.

A further result using the background questionnaire data controlling for experimental effects found that GRE Analytical and Mathematical scores, mathematics grade point averages, number of high school and college mathematics courses taken, and the percentage of working time devoted to programming were all found to correlate to various improved error and/or time performance measures.

The following sections serve as an introduction to the layout of the remainder of this dissertation with brief descriptions of each point of interest along the way.

1.1 Overview

The motivation for and contents of each chapter are discussed in the following sections, summarized here. The dissertation may be considered as consisting of two basic parts, with Part I covering the **Xbro***wser* system, and Part II describing the first major empirical study of parallel program comprehension, performed using **Xbro***wser*.

Part I: The Xbro*wser* System

Part I begins with two chapters reviewing work related to its design and implementation. Chapter 2 on Protocol Analysis presents an introduction to this technique frequently used in empirical user interface research. Chapter 3, Hypertext Issues and Computer-Assisted Learning, discusses work related to the other features of the **Xbro***wser* system. Finally, Chapter 4 presents the **Xbro***wser* system, its component programs, and plans for its future.

1.2 Protocol Analysis

Chapter 2 discusses analysis of both verbal and behavior-based protocols, as well as some existing protocol analysis systems somewhat comparable to **Xbro***wser*, such as Fisher's Protocol Analyst's Workbench [Fis87, Fis91] and the hypertext Writing Environment described by Smith, Smith, and Kupstas [SSK92].

1.3 Hypertext Issues and Computer-Assisted Learning

Chapter 3 presents issues related to other hypertext systems and computer-assisted learning, including an analogous system using another hypertext typesetting language based on the SGML-derived HyTime and MHEG standards [Mar92]: the SGML-MUCH project [ZR93], and how various features of hypertext can be used to support computer-assisted instruction. The "mastery learning" model recommended by Egan [Ega88] to reduce extraneous subject variance through thorough training is used as an example.

1.4 Xbro*wser*: Hypertext and Protocol Analysis for User Interfaces

In Chapter 4 I discuss **Xbro***wser*, a protocol analysis tool for displaying high quality hypertext documents and recording subject responses and system interactions. This

tool was used to present experimental materials and record response times and answers to problems in the study reported in Chapter 7. The three areas of research discussed in Chapters 2 and 3 contributed to the design of **Xbro***wser*: protocol analysis tools such as Fisher's Protocol Analyst's Workbench [Fis87, Fis91] and the protocol analysis instrumentation approach used in the hypertext Writing Environment described by Smith *et al.* [SSK92], hypertext systems such as the analogous SGML-MUCH system [ZR93], and computer-assisted learning issues such as the "mastery learning" technique as described by Egan [Ega88]. The origins of **Xbro***wser* in the pilot studies for the study in Chapter 7 are presented, together with details of the three current components: **xbro** – a hypertext program that displays high-quality typesetting display-device-independent files produced by TEX or LATEX and records a protocol trace of subject interactions with the document, **catevents** – a program that translates the protocol trace event record into formatted data suitable for post-analysis with awk or perl, and **xevents** – a prototype of an event record interactive analysis and replay tool.

Features that distinguish **Xbro***wser* from other protocol analysis systems include its integrated support for computer-assisted learning techniques, its support for direct author-controlled event-level subject protocol recording, and its applicability to multiple domains. The first factor is useful in reducing extraneous experimental error by using appropriate training techniques, the second allows experimenters to deal with many more subjects than might otherwise be possible, and the last represents the flexibility and capability of the system to function in many venues.

Xbro*wser* is distinguished in the area of hypertext and computer-assisted learning by its focus on high-quality typesetting and a simple text-based hypertext model that enhances the ability of hypertext authors to rapidly produce complex documents useful in computer-mediated training and testing. The familiarity and popularity of the TEX and LATEX typesetting model and the focus of the **Xbro***wser* system on core hypertext issues while not precluding more ambitious extensions makes the system simpler and more practical than might otherwise be the case.

Part II: An Empirical Study of Parallel Program Comprehension

Chapters 5 and 6 serve as general introductions to two of the areas of work specifically related to this experiment: empirical studies of sequential programming, and parallel programming and dependence analysis, respectively. In Chapter 7 I then

present the second major contribution of this research, the extensive empirical study of dependence analysis in parallel program comprehension implemented using the **Xbro***wser* system.

1.5 Empirical Studies of Programming

I first give an introduction to empirical studies of sequential programming and programmer variance in Chapter 5 that discusses related work influential in the design of the experiment reported here. In addition to work generally useful for empirical design, this includes in particular studies on the effects of control-flow and data-flow in programming, and studies that attempt to relate subjects' backgrounds to programming performance. In the former category the works most similar to this research are Weiser's studies using slicing [Wei79], and Lyle's studies using slicing and dicing [Lyl84]. Iselin's study is also similar in the choice of dependent variables [Ise88]. In the latter category the most similar work is that of Oman, Curtis, and Nanja on debugging semantic errors [OCN89], although my experimental design in this particular category is also similar to that of Gowda and Saxton [GS89] in terms of background information collected.

1.6 Parallel Programming and Dependence Analysis

Chapter 6 provides some of the motivation for parallel programming, and how dependence analysis is useful for some of its most basic inherent problems. For researchers deeply involved in an area as we are with parallel programming, it is easy to forget that there are appreciable segments of the research audience who may have had little exposure to these topics: I present this as a quick and hopefully painless introduction to the rudiments of parallel programming, and to the fundamentals of dependence analysis that underly the choice of the empirical variables used in the study presented in Chapter 7.

1.7 An Empirical Evaluation of Dependence Analysis in Parallel Program Comprehension

Chapter 7 presents a study performed with the **Xbro***wser* system, in which twenty-six subjects learned the basics of parallel programming and dependence analysis and worked twenty-four randomized problems covering several experimental factors.

Results are presented for severity of errors made by subjects, length of time subjects took to comprehend transformed programs, and individual differences that significantly correlated to these error and time performance measures after controlling for other significant results.

The results of this study as previously alluded to, show that for the prototype interface graphical annotation of program source with dependence information used, the time required to correctly comprehend the detailed and specific results of parallelizing loops was improved. The similarity of this prototype to what is done in the ParaScope parallel programming environment [Che93, CCH+88a, CCH+88b] provides support for the value of the annotation approach in general and the specific method used by ParaScope in particular. In addition, evidence was found that various types of dependences affect both time required for successful comprehension and the severity of errors occurring in the attempt.

The alternative to traditional dependence analysis investigated, derived from a simpler data-flow model, seemed to be no better than the more complete traditional method, nor were parallel loop transformations more difficult to deal with than equivalent sequential loop transformations on the particular task used.

Finally, the background questionnaire data, when controlling for experimental effects, indicated that GRE Analytical and Mathematical scores, mathematics grade point averages, number of high school and college mathematics courses taken, and the percentage of working time devoted to programming were all correlated to various improved error and/or time performance measures.

1.8 Summary, Future Research, and Conclusions

Finally, I reprise in Chapter 8 the conclusions reached and discuss suggested directions for future research for both the **Xbro***wser* system and empirical studies of parallel programming.

Two appendices are also included that describe the experimental materials and statistical results in far greater detail.

Part I

The Xbro*wser* System

This section of the dissertation discusses the **Xbro***wser* hypertext system for protocol analysis. The **Xbro***wser* software, described in Chapter 4, will be made freely available over the Internet. For more information, contact the author by email at `bro@cs.rice.edu`, or consult the World Wide Web page:
`http://www.cs.rice.edu/~bro`.

Chapter 2

Protocol Analysis

Computers have been used to collect and analyze detailed transcriptions of human behavior called *protocols* for some time. These protocols – including data from introspection, thinking aloud, annotation of videotape or audiotape recordings, and output of instrumented programs – constitute a rich resource of empirical data suitable for further analysis. Most systems for protocol analysis process protocols recorded by some other agency and are not integrated into a single framework that includes both recording and analysis. In this chapter I describe some of the research on protocol analysis and protocol analysis support tools related to the **Xbro***wser* system.

Protocol analysis refers to the study of detailed records of human behavior using analytical methods described in more detail below. These records may come in many different forms, and many different methods may be applied to them. Typically, raw protocols may be recordings of either verbal[1] statements or observable behavior; in the latter category, the recording medium may be field notes, auditory, visual, or other mechanical recordings. Verbal and behavioral protocols both may have advantages and disadvantages as summarized in Table 2.1. The perceived disadvantages of verbal protocols include that they are "soft" data: usually consisting of subjective statements, prone to subjective interpretation, produced by verbalization processes that may interfere with the desired object of study, and possibly requiring substantial expertise in psychology to interpret. The advantages of verbal protocols are that they can provide a rich insight into internal thought processes when interpretation is available and can collect information that may not otherwise be observable. The disadvantages of behavioral protocols are that they usually cannot expose internal states of subjects, thereby missing possible internal causation clues, that some observation techniques such as obvious videotaping may perturb subjects, and that the

[1] "Verbal" protocols are typically considered to include both spoken and written records, while verbal recordings in which content is ignored in favor of data such as timings of utterances are really behavioral protocols. [ES93] On the other hand, non-verbal behavior communicative behavior may need to be treated as a "verbal" protocol. [SJS89]

Possible Disadvantages	
Verbal	**Behavioral**
subject-subjective and processed	may ignore internal processes
verbalization usually intrusive	*observation* sometimes intrusive
interpretation often difficult	interpretation sometimes difficult
specialist interpretation may be required	sometimes harder to record
huge data size	"keystroke"-level produces huge data size
Possible Advantages	
Verbal	**Behavioral**
can capture internal processes	usually "hard" data
can be rich, deep data	*observation* can be made imperceptible
	"event"-level (or higher) can reduce data
	data can be simple to interpret
	data can be simple to collect/extract

Table 2.1 Comparison of Verbal and Behavioral Protocols

protocol may be difficult to record (as when the desired behavior to collect requires customized software for each study, or is difficult to evoke). The advantages of behavioral protocols include that they produce "hard" data: specific detailed sets of information that are simple to collect and interpret, that the process of observation may be made imperceptible to subjects by concealment, program instrumentation, *etc.*, and that selection of the grain of recording can reduce extraneous data in the raw protocol. [ES93, Fis91, SJS89]

2.1 A Brief History of Protocol Analysis

Ericsson and Simon [ES93] describe a history of protocol analysis in considerable detail, summarized here. Before the availability of practical methods of sound recording, the only way to record verbalizations consisted of written transcription, typically consisting of field notes that of necessity summarized rather than detailed the spoken record. The similarity of this type of data to the "introspective" method of psychological analysis is marked, and shares many of the drawbacks that led that method to fall into disfavor.

One early analysis of a think-aloud protocol was performed by Watson in 1920, while studies in the 1920s, 1930s, and 1940s continued to contribute to the methodology. Even as late as 1965, DeGroot used as raw data written notes of a think-aloud

protocol to analyze the selection of movements in chess games, using a theoretical framework proposed by Selz in 1913. As primitive phonographic recorders using wax cylinders and disks were replaced with wire-recorders and, particularly after World War II, tape-recorders, the ability to transcribe and analyze verbalizations was greatly improved. The primary bias of the note-taker that could not be addressed except with the use of multiple note-takers was removed, leaving the secondary bias of the analyst, easily addressed because of the existence of an indisputable physical record that could be re-analyzed at will.

Computers also provided new opportunities for protocol analysis, both as analytical and modeling tools and as sources of behavioral protocols. The capabilities of computers and the cognitive demands of using and particularly programming computers made them a rich source of data and tools for protocol analysis. Examples of computer automation of techniques for abstracting and analyzing protocols include the Protocol Analysis Systems PAS-I (1971) and PAS-II (1973) of Waterman and Newell [WN71, WN73]. The founding of the Xerox Palo Alto Research Center (PARC) in 1970 also led to exploration of the study of human factors in the use of computers. This center, discussed by Card, Moran, and Newell [CMN83], has served a fundamental role in many areas of human-computer interaction research.

The work of Card, Moran, and Newell also provided the Keystroke and GOMS[2] models [CMN80, CMN83], frameworks for analysis of behavioral protocols augmented by verbal information. One feature of the GOMS model was its acknowledgment of the value of various grains of analysis, including Keystroke, Argument, Functional, and Unit-Task Levels in order of increasing abstraction. The methodology they used in applying GOMS techniques to protocols consisting of time-stamped keystrokes coupled with symbolic actions produced from think-aloud and video protocols inspired other researchers, including J. Smith, D. Smith, and Kupstas [SSK92] in work involving the Writing Environment described in the next section.

The process of modern protocol analysis consists of several overlapping stages, ranging from the preliminaries of recording and transcribing raw protocols, to segmentation and encoding of protocols into more abstract forms, to data extraction and numerical or logical analyses.[3] A typical verbal protocol analysis might start with a

[2]goals, operators, methods, and selection rules

[3]This model is simplified to allow comparison of somewhat disparate protocol analysis systems. Ericsson and Simon [ES93], Fisher [Fis91], or Sanderson, James, and Seidler [SJS89] are useful references for a fuller picture.

tape recording of a subject, transcribe the tape into text, segment the text into usable phrases or sentences, encode the phrases in symbolic terms, and then use predicate calculus to study the encoding. A behavioral protocol analysis could begin with an instrumented program that produces a record of user-related events that could then be transcribed as plain text for automated segmentation and encoding with filter programs that could also extract specific data for statistical analysis. In this model of protocol analysis, these stages represent:

- **recording** an original raw protocol record.
- **transcription** of rawest protocol into a format usable for analysis.
- **segmentation** of a transcription into units suitable for higher-level encoding.
- **encoding** : symbolic manipulation including "naming" of segmented protocols.
- **data extraction** of specific data to be analyzed, such as encoding-symbol frequencies, time between high-level events, comparison of achievements to ideal situations, *etc.*
- **analysis** of data by numerical, statistical, logical, or other methods

The lack of numbering is intentional: protocol analysis is frequently a non-linear process, and stages typically overlap, recurse, and supplant each other. Fisher describes the chaotic state of the current art: "Protocol analysis is a largely ill-defined, difficult to learn task. There are no cookbook methodologies available, analogous to statistics how-to books, to aid the researcher. In fact, even the most comprehensive treatise on the subject[4] includes only sketchy instructions on the practical application of protocol analysis techniques." [Fis91], p. 3

Procedural issues involving the use of protocol analysis systems include:

- **domain generality** : How hard is it to design and implement a given study assuming the type of protocol supported is adequate? How applicable is the system to other problem domains?
- **data size** : What quantity of low-level data must be processed?
- **automation** : Are automatic processes available to deal with data and can they be monitored and corrected?
- **transportability** : Is some type of intermediate output produced that may be applied to using other systems?

[4]Referring to the 1984 edition of [ES93], this is still essentially correct for the 1993 edition.

2.2 Representative Protocol Analysis Systems

As previously mentioned, early forms of protocol analysis used "think aloud" protocols to record thoughts of subjects either as they occurred, in response to prompting, or retrospectively, using written field notes, and later, audio or video recordings to store the resulting protocol. The availability of useful devices including computers to assist in this process revolutionized protocol analysis, both by providing additional sources of data such as extensive instrumented records of user interactions and by enabling automatic and semi-automatic processing of previously recorded protocols, thereby extending the abilities of protocol analysts to deal with sometimes overwhelming amounts of data. Beside the historical perspective provided by Ericsson and Simon on (mostly verbal) protocol analysis to the present [ES93], J. Smith, D. Smith, and Kupstas [SSK92] and Fisher [Fis87, Fis91] separately describe several systems designed to assist coding and analysis of protocols ranging from field notes to videotape to instrumented-program logs. Representative systems are discussed in the following sections, and are compared to aspects of the **Xbro***wser* system in Chapter 4. Beside Smith *et al.*'s Writing Environment (WE) and Fisher's Protocol Analyst's Workbench (PAW), these systems include the Protocol Analysis Systems PAS-I and PAS-II of Waterman and Newell[WN71, WN73], Verbal Protocol Analysis (VPA[5]) described by Lueke, Pagery, and Brown [LPB87] and work on SHAPA[6] by Sanderson, James, and Seidler [SJS89].

2.2.1 Protocol Analysis Systems PAS-I and PAS-II

These systems included linguistic and semantic processors that operated on natural language transcripts. PAS-I was designed to produce a program behavior graph (PBG) from manually segmented protocols in a completely automatic fashion, while PAS-II extended those capabilities with externally provided rules that made the system more flexible, and allowed user intervention to rescue the system when processing reached a dead end, or when manual tweaking was required. One strong suit, the ability to process natural language, was also a liability in terms of complicating extension of the system to general problems: each problem would require either extension of

[5]VPA is used in the literature both for this particular system and as a generic abbreviation for "verbal protocol analysis".

[6]*Hemi*-Semi-Automated-Protocol-Analysis, with the first two initials intentionally transposed.

the grammar and vocabularies used or a general solution to natural-language parsing. PAS-I and PAS-II support the use of predicate calculus for encoding and analysis.

2.2.2 "Verbal Protocol Analyzer": VPA

VPA uses interactively updated menus of tag categories to segment protocols and build a model interface to meet system-suggested needs; VPA is then used to validate improvements in the new interface using the model. The initial set of menus must be designed beforehand.

2.2.3 "*Hemi*-Semi-Automated-Protocol-Analysis" : SHAPA

SHAPA is explicitly designed to analyze both verbal and non-verbal protocols and assist in developing encoding tags, using the encoding, collecting, and summarizing of data. SHAPA supports the use of predicate calculus for encoding and analysis.

2.2.4 Protocol Analyst's Workbench: PAW

Fisher's own Protocol Analyst's Workbench (PAW) system guides analysts in encoding raw protocol data and selecting appropriate analyses [Fis87, Fis91]. She describes an experiment where six subjects were videotaped thinking aloud about two programming problems. She then used PAW to encode the raw data into a series of concepts and operations demonstrated by all subjects in the process of designing programs, and then performed path analysis to uncover mental models. PAW supports the use of predicate calculus for encoding and analysis.

2.2.5 Trigg's videotape analysis tool

Trigg describes a videotape analysis tool later to be extended to audiotape and field notes, produced as part of the Workplace project at Xerox Palo Alto Research Center (PARC). This tool allowed researchers to produce detailed sets of annotated data streams including text and graphics describing activities captured on videotape [Tri89]. These data streams could then be viewed in various ways and in various levels of detail, and used hypermedia links to constrain layout and presentation.

2.2.6 Experimental Video Annotator: EVA

EVA, an Experimental Video Annotator for symbolic analysis of video data produced by Mackay, runs under UNIX[7] and the X Window System[8] [Mac89]. Using this system, video is digitized and stored in computer-compatible form, and then is integrated into Athena Muse, a language for constructing interactive multimedia [HSA89]. Annotations may be attached to the video, and those annotations can then be filtered and analyzed with other software.

2.2.7 Writing Environment (WE) analysis tools

In contrast to think aloud and non-verbal communicative transcription protocols, the keystroke-level analysis proposed by Card, Moran, and Newell used recorded timestamp information as part of the GOMS models [CMN83]. Much of the work in this direction has been theoretical, describing possible interaction sequences and validating predictions, but Smith *et al.* describe tools and techniques as part of a hypertext production system called the *Writing Environment* (WE) that lend themselves to the underlying empirical schema, but at a higher level of granularity [SSK92]. Instead of recording keystroke-level information, an instrumented version of WE produces a protocol recorded by program instrumentation based on unit *actions*, such as menu selections or text entry, rather than underlying component mouse or keyboard activity. These protocols may then be replayed, filtered, and analyzed as desired, looking for strategic patterns of behavior. The recording method (inserting instrumentation in a program to be studied) is a common approach, but the tools for viewing the resulting protocol are interesting although the system requires more development, particularly to generalize the domain of study, currently limited to users of the Writing Environment itself.

2.3 Conclusions

It is toward this same action-oriented level of protocol analysis used by the Writing Environment analysis tools that our own tool, **Xbro***wser* was designed: capable of detailed records of high-level user interactions but less intrusive in data capture

[7]UNIX is a trademark of AT&T.

[8]The X Window System is a trademark of Massachusetts Institute of Technology.

than videotaping and without the overhead of keystroke-level recording for large experiments. In **Xbro***wser* recorded protocols are the result of users interacting with hypertext documents designed by experimenters. Table 4.1 (previewed below) from Chapter 4 presenting **Xbro***wser* compares support of various protocol analysis systems for certain desirable functions. Solid circles imply strong support for a feature, hollow circles imply at least some support, and blank spaces imply no or weak support. Note that not all systems are designed to support the same set of phases of dependence analysis: this table is intended to be descriptive rather than critical.

Stages	**Xbro***wser**	WE*	PAS	SHAPA	PAW	EVA*
recording	●	●				
transcription	●	●				○
segmentation	●	●	●	●	●	○
encoding	●	●	●	●	●	●
data extraction	●	●	○	○	○	●
analysis	●	●	●	○	●	●
Procedural						
domain generality	●		○	●	●	●
data size	●	●	○	○	○	●
automation	●	●	○	○	●	●
transportability	●	●				○

Key: ● implies strong support, ○ implies moderate support.
* including external filter programs and statistical packages

Preview of Table 4.1:
Comparison of Xbro*wser* to selected protocol analysis systems

Chapter 3

Hypertext Issues and Computer-Assisted Instruction

The **Xbro***wser* protocol analysis and hypertext system presented in Chapter 4 fulfill two distinct roles in human-computer interaction research: it records event-level behavioral protocols of subject interactions with hypertext documents, and it provides a basic set of functions designed to support computer-assisted instruction without enforcing a particular pedagogic model, thereby allowing training and testing using the same system without limiting the design of that training. These two areas constitute work related to the design of the **Xbro***wser* system discussed in the following sections.

3.1 Hypertext Systems

The traditional term "hypertext", coined by Ted Nelson [Nel81], refers to "text ... where there are links between different texts and portions of texts ..." [Ras87]. Though supplemented by the later term "hypermedia" [GSM86, YHMD88] to reflect the ability to include links to graphics, audio, video, *etc.* [TI87], original conceptions of hypertext implicitly included other media [Bus45].

My intent in referring to **Xbro***wser* as *hypertext* rather than *hypermedia* is partly one of modesty: although it is simple to use graphics and external audio or any external program supported by hardware in **Xbro***wser* hypertext documents, the main focus is on simple hypertext links while providing high-quality text formatting. The use of the term hypertext also reflects the basis of the **Xbro***wser* document in the high-quality text layout *dvi*[1]-files created by TEX and LATEX, and a reflection of a desire to improve the longevity of hypertext, cited by Crane as a problem in integrating hypertext into traditional scholarship [Cra87]. Designed originally to create high-quality paper-based output, the device-independent nature of dvi-files allow

[1] **d**e**v**ice-**i**ndependent

ready adaptation to the latest display devices, whether computer displays or printers. As display technology improves, **Xbro***wser* document displays running under the X Window System will improve as well with minimal additional effort.

To focus on the design goals of protocol recording and computer-assisted instruction, extensions to document navigation are left to the author of the **Xbro***wser* documents: in the base system, **Xbro***wser* provides a single level virtual book-mark to be placed by the user. Document designers may augment the basic system to any extent desired by providing menus and graphical displays to guide users, and may prevent digression is sections of the document where it is not desired.

Another feature intended to support computer-assisted instruction is the constraint placed on appearance and placement of the hypertext displays in **Xbro***wser*: by tiling the screen under author control rather than allowing free-form reader placement, and by providing internal session variables that track the current display layout, the author of the hypertext document may be confident of exactly what a reader is seeing without worrying about partial or total obscurement of critical information.

3.1.1 Related Hypertext Systems

One hypertext system with ties to the TEX text formatting system is the **texinfo** package, a product of the GNU Project of the Free Software Foundation. Files prepared using the **texinfo** language can be transformed in one of two ways: into hypertext **info** files that may be viewed using the stand-alone **info** viewer or **GNU emacs** in *info*-mode, or into TEX files suitable for producing a paper document with appropriate cross-references. Despite its use of TEX as an output medium, there is little other resemblance between the **texinfo** system and **Xbro***wser*. When viewing hypertext versions of **texinfo** documents, the text formatting and hypertext elements are primitive at best. From the online documentation distributed with **texinfo** v.3.1/**GNU emacs** v.19.27.1[2]: "Because a Texinfo file must be able to present information both on a character-only terminal in Info form and in a typeset book, the formatting commands that Texinfo supports are necessarily limited."

Another hypertext language that provides primitive text formatting capabilities is HTML (**H**ypertext **M**arkup **L**anguage), used in the World Wide Web (W3) project

[2]From the same source: "Richard M. Stallman wrote Edition 1.0 of this manual. Robert J. Chassell revised and extended it, starting with Edition 1.1."

[BLCG92]. W3 uses viewers such as the dumb-terminal oriented **lynx**[3] or the more capable NCSA Mosaic[4] and Netscape Navigator[5] to display a distributed collection of information nodes, including hypertext/hypermedia documents using HTML. The latter two viewers rely on greater display capabilities such as in the X Windows system and Microsoft Windows environments, and so the text formatting capabilities of HTML are somewhat more powerful that those of **texinfo**. HTML is an SGML (**S**tandardized **G**eneral **M**arkup **L**anguage) DTD (**D**ocument **T**ype **D**efinition), and provides a number of text-like formatting features, including a minimal collection of fonts in varying sizes, titles, paragraphs, and line-breaking. The control of an author over the specific appearance of text is limited, however: like many other hypertext systems, formatting is done on-the-fly and is driven by the specific window size at display time. Resizing a window can completely alter the appearance of the document, and documents may be overlaid by other viewers and windows. As will be seen in the next section, this poses a problem for computer-assisted instruction, which must make assumptions about what pupils can or cannot see. The requirement to support primitive browsers on plain text terminals also restricts the typesetting possibilities of HTML. As the founders of W3 put it: "SGML was not chosen out of any particular aesthetic appeal or inherent cleanliness." [BLCG92], p. 457.

Another emerging hypertext standard, although far less widely used than HTML, is the HyTime project [Mar92]. Like HTML, HyTime is an SGML DTD, although with a larger repertoire of support for multimedia issues, such as time scheduling and synchronization. Textual formatting is not addressed although ODA (**O**ffice **D**ocumentation **A**rchitecture) encodings are being explored.

One application of HyTime is the SGML-MUCH (**M**any **U**sing and **C**reating **H**ypertext) system, again using SGML and, in principle, HyTime as hypertext document interchange and formatting languages [ZR93]. Like **texinfo**, SGML-MUCH can export hypertext documents for use with external text formatters. The degree of text formatting available while viewing hypertext is, like both **texinfo** and HTML, a *browser*-driven problem. No more than plain text is guaranteed, although with some hierarchical organization. Also like **texinfo** and the HTML systems, no author control or knowledge over the field of view seen by readers is provided. These are

[3]Available from the Distributed Computing Group of the University of Kansas.

[4]Developed at the National Center for Supercomputing Applications at the University of Illinois in Urbana/Champaign.

[5]Netscape Navigator is a trademark of Netscape Communications Corporation.

significant limitations for computer-assisted instruction, as will be seen in the next section.

Hypertext is used in **Xbro***wser* to good effect as a schema for document navigation, allowing intelligent response because the system knows what users have seen and done and is able to interactively control where they should be able to go. The general-purpose hypertext design of **Xbro***wser* supports a rich ability to fulfill many needs, including user-interface prototyping and computer-assisted learning. It is in this combination of protocol analysis, hypertext with extensive text layout tools, and support for computer-assisted learning that **Xbro***wser* is unusual.

3.2 Computer-Assisted Learning

Computers provide two components useful in enhancing the learning process: first, they act as a *medium*, displaying desired instructional information in varying ways. Second, computers may be used to *mediate* in the instructional process, serving as agents to administer lessons for an instructor.

A central problem in this second role of computer-assisted learning is support for "authoring": the process of writing and assembling lessons to be applied by the computer as tutor. Müldner and Elammari [ME92] briefly review several authoring systems including Hypercard, Toolbook, Quest, IconAuthor, Tencore, and their own OBJECTOR. This system features multiple windows with a rich user interface and learner-modifiable hypertext, and uses C++ [Str91] as its input language.

An extensive example of the use of ABASE – a Hypercard-based interactive tutorial program – in a specific information domain is given by Li, Rovick, and Michael [LRM92]. They describe several problems and tutorials in acid/base regulation used by first-year medical students; each is implemented as a Hypercard stack. These feature hypertext, animated summaries, graphical testing routines, and an interactive dictionary.

Hypertext is only one way that computers can fill the role of an instructional medium. Badre *et al.* [BBMS92] discuss the use of program visualization systems [Mye90] in general to assist in teaching computer science. They also discuss the use of XTango, an algorithm animation system descended from Tango by John Stasko [Sta90]. Stasko and Patterson [SP91] present a classification system for program visualization tools in four dimensions – aspect, abstractness, animation, and automation – that apply to other computer-media as well. **Xbro***wser* has sufficient capabilities

to fulfill several levels of most of these categories, with animation being the sole exception unless external programs are used, as is possible.

3.2.1 Navigation and Visual Control Issues in Hypertext for Computer-Assisted Instruction

When used as instructional medium, some issues in hypertext become even more important. De La Passardiere and Dufresne [LD92] present a survey of navigational tools used in hypermedia, including overview, local, and fisheye maps, filters, and indices. Other tools such as history trails, footprints, landmarks, and progression cues are more obviously useful in the construction of intelligent lessons: by knowing what a student has seen, the lesson may adapt to present remaining topics in the most constructive way possible. This ability to adapt and respond is critical to the educational function.

The issue of visual control is part of the problem of disorientation inherent in hypertext systems. [LD92, Con86] Readers may easily become lost in complex documents, and without adequate author support for awareness of current document location, the hypertext system must provide mechanisms to orient the reader, when it is the *author* who is most qualified to guide the reader to progress rather than digress.

Xbro*wser* addresses this issue in several ways. The hypertext author may place and test range-marker annotations within the document or test hypertext page numbers to determine the current location, and may keep a trail of where readers have been. Session-specific variables allocated for individual readers can keep a record of what has been seen and accomplished, allowing the author to guide readers onward. The ability to continue and the destination of conventional reading operations such as "next-page" and "previous-page" are always under author control. **Xbro***wser* provides as a primitive a single-level "virtual bookmark" for readers to place and return to within a document. By not forcing a navigation mechanism on authors and by allow authors to control the availability of the primitive bookmark, **Xbro***wser* allows much finer control over digression than is usual in hypertext systems: authors may allow or encourage it, implementing more extensive "stacks" of bookmarks if desired, or may turn off the ability to digress when some portion of the document requires close attention.

Another feature of **Xbro***wser* that is useful in computer-assisted instruction is the control provided over available windowing displays: four combinations are provided, ranging from one to four windows that **tile** the display without overlap. An author is thus guaranteed that if the session variables indicate a particular bit of text is available, then readers may see it. The problem of obscurement of critical information is thus precluded.

3.2.2 "Mastery Learning" : a Computer-Assisted Instruction Paradigm

One approach to training that is particularly applicable when using computers is called automated "mastery learning", as described by Egan [Ega88]. Motivated by a desire to reduce extraneous differences in performance in subjects after training, these techniques use a computer to administer lessons that are :

- intended to be mastered with proficiency by all students,
- are broken into small pieces sequenced so that prerequisite skills are mastered before higher-level skills,
- are followed by tests after each unit of instruction,
- and provide remedial instruction immediately when needed.

These principles were used in the design of instructional material used in the experiment presented in Chapter 7.

The design of **Xbro***wser* facilitates the implementation of methods such as automated mastery learning by providing traditional document structuring features that allow compartmentalization of lessons in familiar "chapter-section-subsection" document models, while allow interactive questioning of readers to determine understanding of a given topic, thereby allowing normal progression or remedial instruction to be offered.

3.3 Conclusion

It may thus be seen that the **Xbro***wser* system is a unique hypertext medium, allowing high-quality text formatting coupled with author control of presentation and progress. Together with its support for protocol analysis, these features make **Xbro***wser* a valuable tool in performing empirical studies of programming.

Chapter 4

Xbro*wser*: Hypertext and Protocol Analysis for User Interfaces

In this chapter I describe **Xbro***wser*, a hypertext system suitable for computer-assisted learning, user-interface prototyping, and empirical event-level protocol analysis studies using features that record subject interactions with hypertext documents prepared using TEX[1] [Knu79] or LATEX [Lam94]. **Xbro***wser* is the product of a confluence of protocol analysis, hypertext, and computer-assisted learning systems, and was used in the extensive study described in Chapter 7. I will discuss **Xbro***wser* and my experiences using **Xbro***wser* in such experiments shortly, but because of the wide variety of related work, some background information is presented first.

4.1 Areas related to Xbro*wser*

4.1.1 Protocol Analysis

Computers have been used to collect and analyze detailed transcriptions of data on human-computer interaction – called *protocols* – for some time. These protocols, whether collected through field notes, annotation of videotape or audiotape recordings, or instrumented programs, provide a rich source of empirical data to be analyzed. Unlike **Xbro***wser*, most systems for protocol analysis are used to process protocols recorded in some other medium and are not integrated in an application that provides general-use hypertext facilities or features applicable to computer-assisted learning.

Early forms of protocol analysis used "think aloud" protocols to record thoughts of subjects either as they occurred, in response to prompting, or retrospectively, using written field notes, and later, audio or video recordings to transcribe the resulting protocol. The availability of useful devices including computers revolutionized protocol analysis, both by providing additional sources of data such as extensive records of user interactions and by enabling automatic and semi-automatic processing of

[1] TEX is a trademark of the American Mathematical Society

recorded protocols to extend the abilities of protocol analysts to deal with sometimes overwhelming amounts of data. Ericsson and Simon provide a useful history of protocol analysis to the present [ES93]; J. Smith, D. Smith, and Kupstas [SSK92] and Fisher [Fis87, Fis91] separately describe several systems designed to assist coding and analysis of protocols ranging from field notes to videotape to instrumented-program logs. Beside Smith *et al.*'s Writing Environment (WE) and Fisher's Protocol Analyst's Workbench (PAW), these include the Protocol Analysis Systems PAS-I and PAS-II of Waterman and Newell [WN71, WN73], Verbal Protocol Analysis (VPA[2]) described by Lueke, Pagery, and Brown [LPB87] and work on SHAPA by Sanderson, James, and Seidler [SJS89].

PAS-I was designed to produce a program behavior graph (PBG) from manually segmented protocols; PAS-II extended its capabilities with externally provided rules that made the system more flexible. VPA uses interactively updated menus of tag categories to segment protocols and build a model interface to meet system-suggested needs; VPA is then used to validate improvements in the new interface. SHAPA is explicitly designed to analyze both verbal and non-verbal protocols and assist in developing encoding tags, using the encoding, and collecting and summarizing data.

Fisher's own Protocol Analyst's Workbench (PAW) system guides analysts in encoding raw protocol data and selecting appropriate analyses [Fis87, Fis91]. She describes an experiment where six subjects were videotaped thinking aloud about two programming problems. She used PAW to encode the raw data into a series of concepts and operations demonstrated by all subjects in the process of designing programs, and then performed path analysis to uncover mental models.

Trigg describes a videotape analysis tool later to be extended to audiotape and field notes, produced as part of the Workplace project at Xerox Palo Alto Research Center (PARC). This tool allowed researchers to produce detailed sets of annotated data streams including text and graphics describing activities captured on videotape [Tri89]. These data streams could then be viewed in various ways and in various levels of detail, and used hypermedia links to constrain layout and presentation.

EVA, an Experimental Video Annotator for symbolic analysis of video data produced by Mackay, runs under UNIX[3] and the X Window System[4] [Mac89]. Using

[2]VPA is used in the literature both for this particular system and as a generic abbreviation for "verbal protocol analysis".

[3]UNIX is a trademark of AT&T.

[4]The X Window System is a trademark of Massachusetts Institute of Technology.

this system, video is digitized and stored in computer-compatible form, and then is integrated into Athena Muse, a language for constructing interactive multimedia [HSA89]. Annotations may be attached to the video, and those annotations can then be filtered and analyzed with other software.

In contrast to think aloud and human-observation transcription protocols, Card, Moran, and Newell proposed a keystroke-level analysis using recorded timestamp information as part of the Keystroke and GOMS[5] models [CMN80, CMN83]. Much of the work in this direction has been theoretical, describing possible interaction sequences and validating predictions, but Smith *et al.* describe tools and techniques as part of a hypertext production system called the *Writing Environment* (WE) that lend themselves to the underlying empirical schema, but at a higher level of granularity [SSK92]. Instead of recording keystroke-level information, WE produces a protocol recorded by program instrumentation based on unit *actions*, such as menu selections or text entry, rather than underlying component mouse or keyboard activity. These protocols may then be replayed, filtered, and analyzed as desired, looking for strategic patterns of behavior.

It is toward this same action-oriented level of protocol analysis that our own tool, **Xbro***wser* was designed: capable of detailed records of high-level user interactions but less intrusive in data capture than videotaping and without the overhead of keystroke-level recording for large experiments. In **Xbro***wser* recorded protocols are the result of users interacting with hypertext documents designed by experimenters.

The model of protocol analysis presented in Chapter 2 consists of several non-exclusive stages:

- **recording** an original raw protocol record.
- **transcription** of rawest protocol into a format usable for analysis.
- **segmentation** of a transcription into units suitable for higher-level encoding.
- **encoding** : symbolic manipulation including "naming" of segmented protocols.
- **data extraction** of specific data to be analyzed, such as encoding-symbol frequencies, time between high-level events, comparison of achievements to ideal situations, *etc.*
- **analysis** of data by numerical, statistical, logical, or other methods

[5]goals, operators, methods, and selection rules

There are also procedural issues concerning the use of protocol analysis systems, including domain generality (how easily different domains of study can use the system, ignoring differences in verbal and behavioral protocols), data size (how much data must be dealt with) automation (whether data may be readily dealt with automatically), and transportability (whether intermediate output is produced that may be applied to using other systems.)

Table 4.1 compares how some of these systems compare to **Xbro***wser* in support for desirable functions. Solid circles imply strong support for a feature, hollow circles imply at least some support, and blank spaces imply no or weak support. Note that not all systems are designed to support the same set of phases of dependence analysis: this table is intended to be descriptive rather than critical.

Stages	**Xbro***wser**	WE*	PAS	SHAPA	PAW	EVA*
recording	●	●				
transcription	●	●				○
segmentation	●	●	●	●	●	○
encoding	●	●	●	●	●	●
data extraction	●	●	○	○	○	●
analysis	●	●	●	○	●	●
Procedural						
domain generality	●		○	●	●	●
data size	●	●	○	○	○	●
automation	●	●	○	○	●	●
transportability	●	●				○

Key: ● implies strong support, ○ implies moderate support.
* including external filter programs and statistical packages

Table 4.1 Comparison of **Xbro***wser* to selected protocol analysis systems

4.1.2 Hypertext Systems

The traditional term "hypertext", coined by Ted Nelson [Nel81], refers to "text ... where there are links between different texts and portions of texts ..." [Ras87]. Though supplemented by the later term "hypermedia" [GSM86, YHMD88] to reflect the ability to include links to graphics, audio, video, *etc.* [TI87], original conceptions of hypertext implicitly included other media [Bus45].

My intent in referring to **Xbro***wser* as *hypertext* rather than *hypermedia* is partly one of modesty: although it is simple to use graphics and external audio or any external program supported by hardware in **Xbro***wser* hypertext documents, the main focus is on simple hypertext links while providing high-quality text formatting. The use of the term hypertext also reflects the basis of the **Xbro***wser* document in the high-quality text layout *dvi*[6]-files created by TEX and LATEX, and a reflection of a desire to improve the longevity of hypertext, cited by Crane as a problem in integrating hypertext into traditional scholarship [Cra87]. Designed originally to create high-quality paper-based output, the device-independent nature of dvi-files allow ready adaptation to the latest display devices, whether computer displays or printers. As display technology improves, **Xbro***wser* document displays running under the X Window System will improve as well with minimal additional effort.

One hypertext system with ties to the TEX text formatting system is the **texinfo** package, a product of the GNU Project of the Free Software Foundation. Files prepared using the **texinfo** language can be transformed in one of two ways: into hypertext **info** files that may be viewed using the stand-alone **info** viewer or **GNU emacs** in *info*-mode, or into TEX files suitable for producing a paper document with appropriate cross-references. Committed to supporting the most primitive terminals, text formatting in **texinfo** is necessarily limited, and the hypertext elements only the most basic.

Another hypertext language that provides primitive text formatting capabilities is HTML (**H**yper**t**ext **M**arkup **L**anguage), used in the World Wide Web (W3) project [BLCG92]. W3 uses viewers such as the dumb-terminal oriented **lynx**[7] or the more sophisticated NCSA Mosaic[8] and Netscape Navigator[9] to display a distributed collection of information nodes, including hypertext/hypermedia documents using HTML. Again, the requirement to support primitive browsers on plain text terminals restricts the typesetting possibilities of HTML, and in windowing versions of viewers, there are the usual problems of obscurement and disorientation inherent in windowed hypertext.

---

[6] **de**vice-**i**ndependent

[7] Available from the Distributed Computing Group of the University of Kansas.

[8] Developed at the National Center for Supercomputing Applications at the University of Illinois in Urbana/Champaign.

[9] Netscape Navigator is a trademark of Netscape Communications Corporation.

Another SGML-derived hypertext proposal – HyTime [Mar92] – is used in the SGML-MUCH (**M**any **U**sing and **C**reating **H**ypertext) system [ZR93]. Like **texinfo**, SGML-MUCH can export hypertext documents for use with external text formatters. The degree of text formatting available while viewing hypertext is, like both **texinfo** and HTML, a *browser*-driven problem. No more than plain text is guaranteed, although with some hierarchical organization. Also like **texinfo** and the HTML systems, no author control over or knowledge about the field of view seen by readers is provided. These are significant limitations for computer-assisted instruction, as will be seen in the next section.

To focus on the design goals of protocol recording and computer-assisted instruction, extended document navigation is left to the author of an **Xbro***wser* document: in the base system, **Xbro***wser* provides a single level virtual book-mark to be placed by the user. Document designers may augment the basic system to any extent desired by providing menus and graphical displays to guide users, and may prevent digression is sections of the document where it is not desired.

Another feature intended to facilitate computer-assisted instruction is a constraint on appearance and placement of the windows displaying hypertext in **Xbro***wser*: by tiling the display under author control rather than allowing free-form reader placement, and by providing internal session variables that track the current display layout, the author of the hypertext document may be confident as to exactly what a reader is seeing without worrying about partial or total obscurement of critical information.

Hypertext is used in **Xbro***wser* to good effect as a schema for document navigation, allowing intelligent response because the system knows something of what users have seen and is able to interactively adjust where they should be able to go. The general-purpose hypertext design of **Xbro***wser* supports a rich ability to fulfill many needs, including user-interface prototyping and computer-assisted learning. It is in this combination of protocol analysis, hypertext with extensive text layout tools, and support for computer-assisted learning that **Xbro***wser* is unusual.

4.1.3 Computer-Assisted Learning

Computers provide two components useful in enhancing the learning process: first, they act as a *medium*, displaying desired instructional information in varying ways. Second, computers may be used to *mediate* in the instructional process, serving as agents to administer lessons for an instructor.

A central problem in this second role of computer-assisted learning is support for "authoring": the process of writing and assembling lessons to be applied by the computer as tutor. Müldner and Elammari [ME92] briefly review several authoring systems including Hypercard, Toolbook, Quest, IconAuthor, Tencore, and their own OBJECTOR. This system features multiple windows with a rich user interface and learner-modifiable hypertext, and uses C++ [Str91] as its input language.

An extensive example of the use of ABASE – a Hypercard-based interactive tutorial program – in a specific information domain is given by Li, Rovick, and Michael [LRM92]. They describe several problems and tutorials in acid/base regulation used by first-year medical students; each is implemented as a Hypercard stack. These feature hypertext, animated summaries, graphical testing routines, and an interactive dictionary.

Hypertext is only one way that computers can fill the role of an instructional medium. Badre *et al.* [BBMS92] discuss the use of program visualization systems [Mye90] in general to assist in teaching computer science. They also discuss the use of XTango, an algorithm animation system descended from Tango by John Stasko [Sta90]. Stasko and Patterson [SP91] present a classification system for program visualization tools in four dimensions – aspect, abstractness, animation, and automation – that apply to other computer-media as well. **Xbro***wser* has sufficient capabilities to fulfill several levels of most of these categories, with animation being the sole exception unless external programs are used, as is possible.

When used as instructional medium, some issues in hypertext become even more important. De La Passardiere and Dufresne [LD92] present a survey of navigational tools used in hypermedia, including overview, local, and fisheye maps, filters, and indices. Other tools such as history trails, footprints, landmarks, and progression cues are more obviously useful in the construction of intelligent lessons: by knowing what a student has seen, the lesson may adapt to present remaining topics in the most constructive way possible. This ability to adapt and respond is critical to the educational function.

The issue of visual control is part of the problem of disorientation inherent in hypertext systems. [LD92, Con86] Readers may easily become lost in complex documents, and without adequate author support for awareness of current document location, the hypertext system must provide mechanisms to orient the reader, when it is the *author* who is most qualified to guide the reader to progress rather than digress.

Xbro*wser* addresses this issue in several ways. The hypertext author may place and test range-marker annotations within the document or test hypertext page numbers to determine the current location, and may keep a trail of where readers have been. Session-specific variables allocated for individual readers can keep a record of what has been seen and accomplished, allowing the author to guide readers onward. The ability to continue and the destination of conventional reading operations such as "next-page" and "previous-page" are always under author control. **Xbro***wser* provides as a primitive a single-level "virtual bookmark" for readers to place and return to within a document. By not forcing a navigation mechanism on authors and by allow authors to control the availability of the primitive bookmark, **Xbro***wser* allows much finer control over digression than is usual in hypertext systems: authors may allow or encourage it, implementing more extensive "stacks" of bookmarks if desired, or may turn off the ability to digress when some portion of the document requires close attention.

Another feature of **Xbro***wser* that is useful in computer-assisted instruction is the control provided over available windowing displays: four combinations are provided, ranging from one to four windows that **tile** the display without overlap. An author is thus guaranteed that if the session variables indicate a particular bit of text is available, then readers may see it. The problem of obscurement of critical information is thus precluded.

One approach to training that is particularly applicable when using computers is called automated "mastery learning", as described by Egan [Ega88]. These techniques use a computer to administer lessons that are broken into small pieces sequenced so that prerequisite skills are mastered before higher-level skills. Then tests are made of the acquisition of those skills, with remedial instruction if needed. These principles were used in the design of instructional material used in the experiment presented in Chapter 7.

4.2 The Genesis of Xbro*wser*

The need for **Xbro***wser* was recognized during the design of an experiment on empirical analysis of dependence analysis in parallel programming described in Chapter 7. As part of that design process several pilot studies were performed to identify experimental issues and variables. In the second pilot study, four subjects were videotaped using written materials on parallel programming to work 24 problems testing com-

prehension of effects of reversing and parallelizing loop transformations; the materials contained colored flags that made it possible to code time sequence information from the videotape as subjects completed various instructional sections and individual problems. Analysis of the results made it clear that these time values could be valuable, but that the process of videotaping and analysis would be problematic on the scale required to demonstrate projected effects. However, the written materials prepared from device-independent dvi-files produced by LaTeX worked well other than the need for some mechanism to record user interactions.

The next step was inspired by an existing dvi-file previewer running under the X Window system called XTEX, begun by Dirk Grunwald while at M.I.T. using a library of routines written by Chris Torek at the University of Maryland [GT90]. The XTEX previewer provides a primitive hypertext capability for references using dvi-files as input, produced simply by including an "xtex" LaTeX style option. This observation led to the initial design of a full hypertext program running under X Windows that used annotated dvi-files and also recorded user interactions. Once refined this design became the **Xbro***wser* hypertext system for protocol analysis.

4.3 The Xbro*wser* system

Xbro*wser* is an extensible set of tools for protocol analysis that consists of two components programs in its current implementation: **xbro**, a flexible hypertext presentation program that uses dvi-files produced by TeX or LaTeX from source provided by experimenters and optionally stores action-oriented protocol records for each user; and **catevents**, a translator that converts compact event records produced by **xbro** into human-readable form suitable for further analysis by filter and analytic tools running under Unix such as awk or perl. Future tools include **xevents**, an interactive event record replay and filter-creation program currently in prototype.

4.3.1 xbro: hypertext viewer and protocol recorder

The first step in using the **Xbro***wser* system is to prepare a hypertext dvi-file for display using TeX or LaTeX; a style option called "xbro" provides an interface using macros to access the basic hypertext features. Fig. 4.1 is a simple example of a LaTeX file that produces an **Xbro***wser* document with two hypertext buttons, using the **xbro** style option: the source produces two displays, the relevant portions of which are shown, and writes two custom events into the protocol event record,

```
\documentstyle[xbro]{article}
\def\firstpage{1}
\def\secondpage{2}
\xbroevent{\firstpage}
\begin{document}
This is a \xbrobutton{button}{forward()} that ...

back to this page $\Leftarrow$
\clearpage
... leads to here.

\xbrobutton{This button }{backward()} takes you $\Leftarrow$
\xbroevent{\secondpage}
\end{document}
```

Sample Xbro*wser* LaTeX file

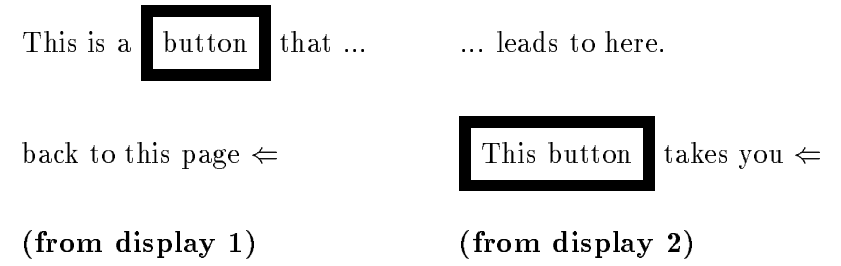


Figure 4.1 Sample Xbro*wser* LaTeX file and resulting **xbro** displays

one at the beginning when the first page is displayed, and one when the second page finishes displaying. The **xbro** program displays pages from the hypertext dvi-file in one to four windows called displays tiling the entire screen, looking for "special" commands embedded in the dvi file. These **xbro** directives belong to five categories: **actions** (including conditional actions based on the value of session variables) that are executed when encountered, such as tiling the screen in a particular way, setting session variables, opening a new hypertext dvi-file, going to a particular page, *etc.*; **action-oriented buttons** that display text or graphics within a frame (hollow when disabled) to indicate the button may be selected with the mouse to execute associated actions; **status-oriented buttons** such as multiple-choice or option-selection buttons, *etc.*; **display-oriented indicators** that selectively conceal or highlight text or depict numeric values as bar graphs based on session variable values; or **editors**

that use X Windows standard editing capabilities to allow users to enter text stored in session variables. Session variables and the current document and location can be saved in user-specific context files for later retrieval, allowing subjects to quit and resume at will. The current implementation of **Xbro***wser* uses standard Athena widget editors; future versions will use Motif [10] widget editors or XView[11] text subwindows, allowing a more flexible user interface when filling in multiple editors.

Users of **xbro** access hypertext dvi-files either directly from the command line or by using a login procedure that simplifies administration of experiments using the **Xbro***wser* system. Fig. 4.2 shows the appearance of **xbro** after a user has logged into **Xbro***wser* using the dialog on the left; for consistency, the login dialog remains visible whenever a single display is active, as in this figure. The login dialog contains a scrollable output window, standard X Windows editors for entering a subject's **Xbro***wser* user name and password to access specific documents used by that subject, a quit button, a help button that provides context-sensitive help in the login dialog output window, and a button to complete the login process. The **Xbro***wser* user name and password are used to verify access to files named in a simple **Xbro***wser* database, protecting subject confidentiality; those files specify for each subject an initially assigned hypertext document, a context file to maintain location and session variables so that subjects may quit and resume at will, and an event file that records interactions of the subject with the system and documents. Both context and event files are optional; only a suitable dvi-file is required. Multiple experiments can be supported within either a single database or multiple databases stored in the UNIX file system.

Once the document and most recent context of a subject is accessed, the associated display arrangement is presented. Fig. 4.2 shows an introductory page in the single display on the right, with the pre-defined standardized "control-panel" at the top of the display produced by LaTeX macros defined in the provided "xbro" style option; other such macros allow creation of the various actions, buttons, indicators, and editors, or insert invisible "milestone" events that appear in the output event record when the associated display is presented. Standard control-panel buttons allow quitting the **xbro** session (disabled for demonstration purposes in Fig. 4.2 as can be seen by its hollow button box), visiting a table of contents (likewise disabled),

[10] OSF/Motif is a trademark of the Open Software Foundation, Inc.

[11] XView is a trademark of Sun Microsystems, Inc.

Figure 4.2 **Xbro***wser* login window from **xbro** with single display

paging backward and forward through the document where appropriate, "pushing" a virtual book-mark at the current page, or "popping" back to a previously placed book-mark if any. Here, access to the table of contents is disabled until the introduction to **Xbro***wser*'s features is completed, and no book-mark exists to be popped. Whenever the next-page function is disabled, the mouse cursor becomes a question-mark to remind users that some action needs to be performed to re-enable next-page. The mouse cursor also changes to a standard watch whenever **xbro** is occupied with system tasks and is ignoring user input: earlier versions that queued user input in such circumstances for later processing proved problematic. As an alternative to clicking on embedded buttons, subjects can use keyboard equivalents for many control-panel functions. Statistics on the numbers of buttons, edit events, and keyboard commands used are all maintained by **xbro** in the event record, allowing statistical profiling of individual interaction styles. The event record also contains all editing text events and a trace of advancement through the document on a page-by-page or symbolic-milestone basis as decided by the experimenter.

In addition to these action-oriented buttons, display-oriented indicators may be used to conceal or invert text according to the status of session variables, progression bar-gauges may be used to show numeric values graphically, and editors may be used to input values to be stored in variables and the event record. Fig. 4.3 is a four-display tiled screen that was used as part of the training process for the experiment presented in Chapter 7. The display in the upper right contains examples of both indicator types, whose status depends on the contents of the 30 work sheet editors in the lower left. In this example, the Affected Set Name in the editor for Step D.1 is "second" rather than "fourth"; concealment indicators in the bottom third of the upper right display indicate this answer is incorrect and what the correct answer is. At the bottom of the upper right display, inverting indicators show that the answers in the work sheet editors are correct through Step C; once the Affected Set Name editor is corrected the indicators for Steps D.1 to D.3 will invert and the "Next" page button will be enabled, allowing the subject to continue to the next screen. In this example, the remaining work sheet editors are correct, but the third multiple-choice button answer in the answer sheet in the lower right display is incorrect; the subject must click on the incorrect "Yes" button to re-enable the "No" alternative and then select it to complete this programmed-instruction review of the first training problem.

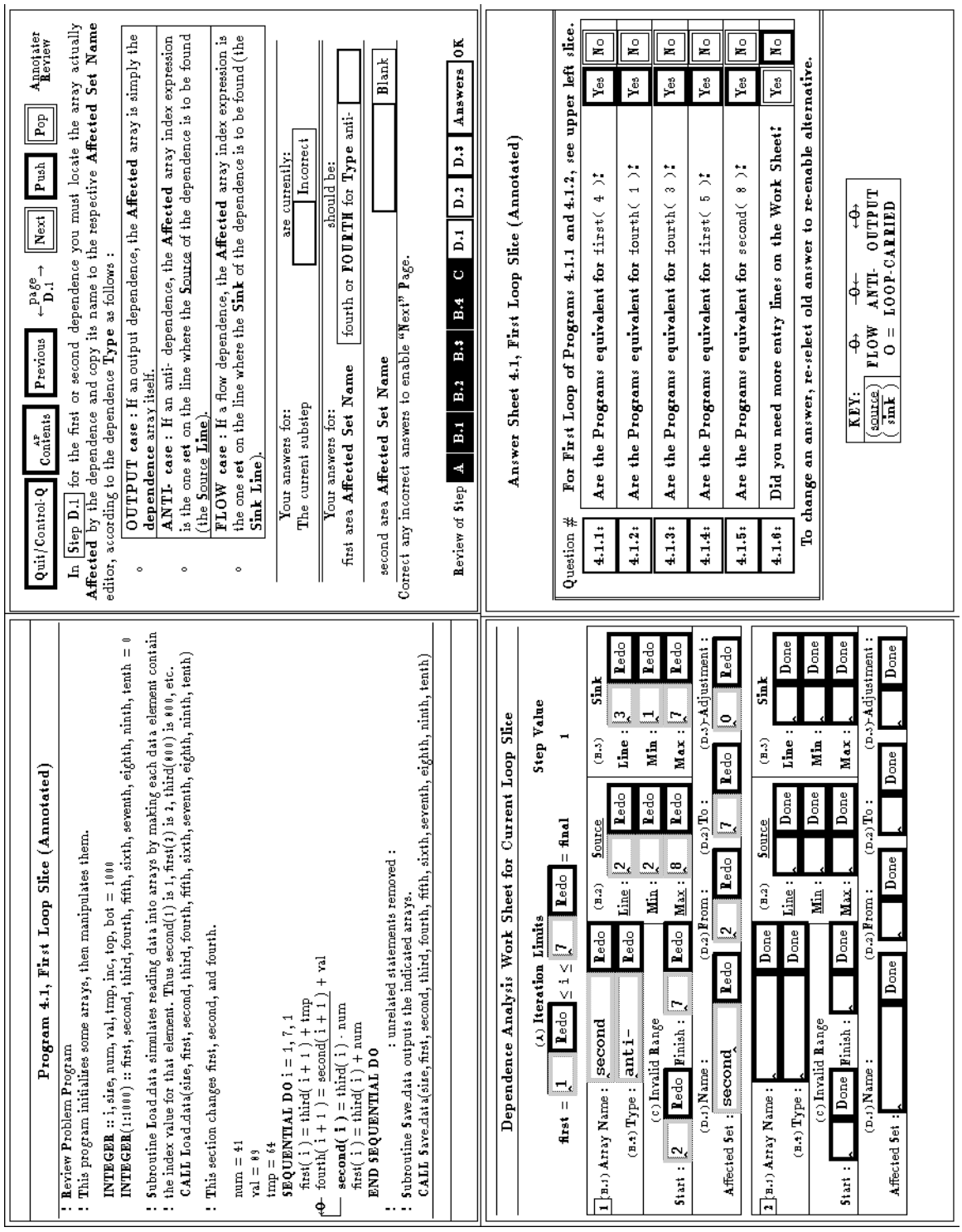


Figure 4.3 Complex **Xbro***wser* example from **xbro** with four displays

4.3.2 catevents: simple protocol event display

The events records stored by **xbro** are in an uncompressed binary form. The **catevents** program was created to convert event records into human-readable text; filter programs such as awk or perl can then scan this text for further processing. Additional **Xbro***wser* tools under development use whichever of the original event record file or the output of **catevents** is more convenient.

Fig. 4.4 is a partial sample of the output of **catevents** for a subject completing an introduction document to **Xbro***wser*. In order, the events depicted show the subject logged in to **Xbro***wser* on an X-terminal called "typical.cs.rice.edu" using a CPU-server called "mythical"[12], triggered an initial embedded milestone called "FIRST

```
# fields:
# elapsed time since last event
# |     wall-clock time since first xbro login
# |     |   event name and assigned value
# |     |   |              |   timestamp            display:page1/2/3/4
# |     |   |              |   |                            |
# V     V   V              V   V                            V
#-------------------------------------------------------------------------
  0 :   0 :  EVENT START : -3 :Mon Oct 11 11:09:36 1994:VERSION 3.0
  0 :   0 :   HOST EVENT : -5 :Mon Oct 11 11:09:36 1994:\
Host=:mythical:Display=:typical.cs.rice.edu:0.0:
  0 :   0 :   FIRST GOAL :  0 :Mon Oct 11 11:09:36 1994:
  0 :   0 :  PAGE1 EVENT : -6 :Mon Oct 11 11:09:36 1994:PG=:0:1
125 : 125 :  PAGE1 EVENT : -6 :Mon Oct 11 11:11:41 1994:PG=:1:2
  5 : 130 :  PAGE1 EVENT : -6 :Mon Oct 11 11:11:46 1994:PG=:0:1
  2 : 132 :  PAGE1 EVENT : -6 :Mon Oct 11 11:11:48 1994:PG=:1:2
 88 : 220 :  PAGE3 EVENT : -8 :Mon Oct 11 11:13:16 1994:PG=:4:5:6:7:7:8
  9 : 229 :  PAGE4 EVENT : -9 :Mon Oct 11 11:13:25 1994:PG=:5:6:6:7:7:8:8:9
  6 : 236 :  PAGE2 EVENT : -7 :Mon Oct 11 11:13:32 1994:PG=:3:4:6:7
  3 : 239 :  PAGE1 EVENT : -6 :Mon Oct 11 11:13:35 1994:PG=:2:3
  5 : 244 :  PAGE1 EVENT : -6 :Mon Oct 11 11:13:40 1994:PG=:9:10
 85 : 329 :   EDIT EVENT : -10:Mon Oct 11 11:15:05 1994:firstedit:1:\
This is an example of a multiple-line\n\ edit entry.
 24 : 353 : FINISH INTRO :  1 :Mon Oct 11 11:15:29 1994:PG=:10:11
```

Figure 4.4 Sample **Xbro***wser* output from **catevents**

[12]machine names are fictionalized

awk script

```
# awk example for page tracing and section time summaries:
# traces = page(s+/+/+)-[elapsed time]-page(s+/+/+)...
BEGIN        { printf("> " ); sum = 0 ; total = 0 ; FS=":" ; }
( NF > 6)    { total += $1 ; sum += $1 ; }
/PG=/        { printf("-[%d]-%d", sum, $10) ; sum = 0 ;
               for ( f = NF ; f > 10 ; f -=2 ) printf("+") ; }
( $4+0 > 0 ) { printf("\n>total time to %s was %d seconds.\n", $3, total) ;
               total = 0 ; }
```

awk output

```
>-[0]-1-[125]-2-[5]-1-[2]-2-[88]-5++-[10]-6+++-[6]-4+-[3]-3-[5]-10-[109]-11
>total time to  FINISH INTRO  was 353 seconds
```

Figure 4.5 Short awk example for scanning **catevents** output

GOAL", spent 125 seconds viewing the initial display (0:1) before going to display 1:2, five seconds later went back to 0:1, two seconds later went on again to 1:2, 88 seconds later started a three-display screen with displays 4:5, 6:7, and 7:8, *etc.* The EDIT EVENT shows the content of an editor variable named "`firstedit`" was changed 85 seconds after arriving at display 9:10. Editor contents are session variables that may be used in conditional expressions in the hypertext annotations stored in the dvi-file. The document used in this example required `firstedit` to contain non-blank text before the next-page function was enabled: 24 seconds after entering the text, the subject finished reading the page and proceeded to display 10:11, triggering the "FINISH INTRO" milestone; the total time for this subject to finish the introduction to **Xbro***wser* was just under six minutes.

Milestone names are contained either in an external file maintained by the author of the hypertext document, or may be stored at the beginning of an event file as desired; the external file is used in producing the document so event records and symbols automatically match. The output of **catevents** is designed for use by pattern matching filter programs. For example, Fig. 4.5 shows a simple awk script for scanning **catevents** output that outputs page usage and milestone times.

4.3.3 xevents: prototype replay tool and event filter editor

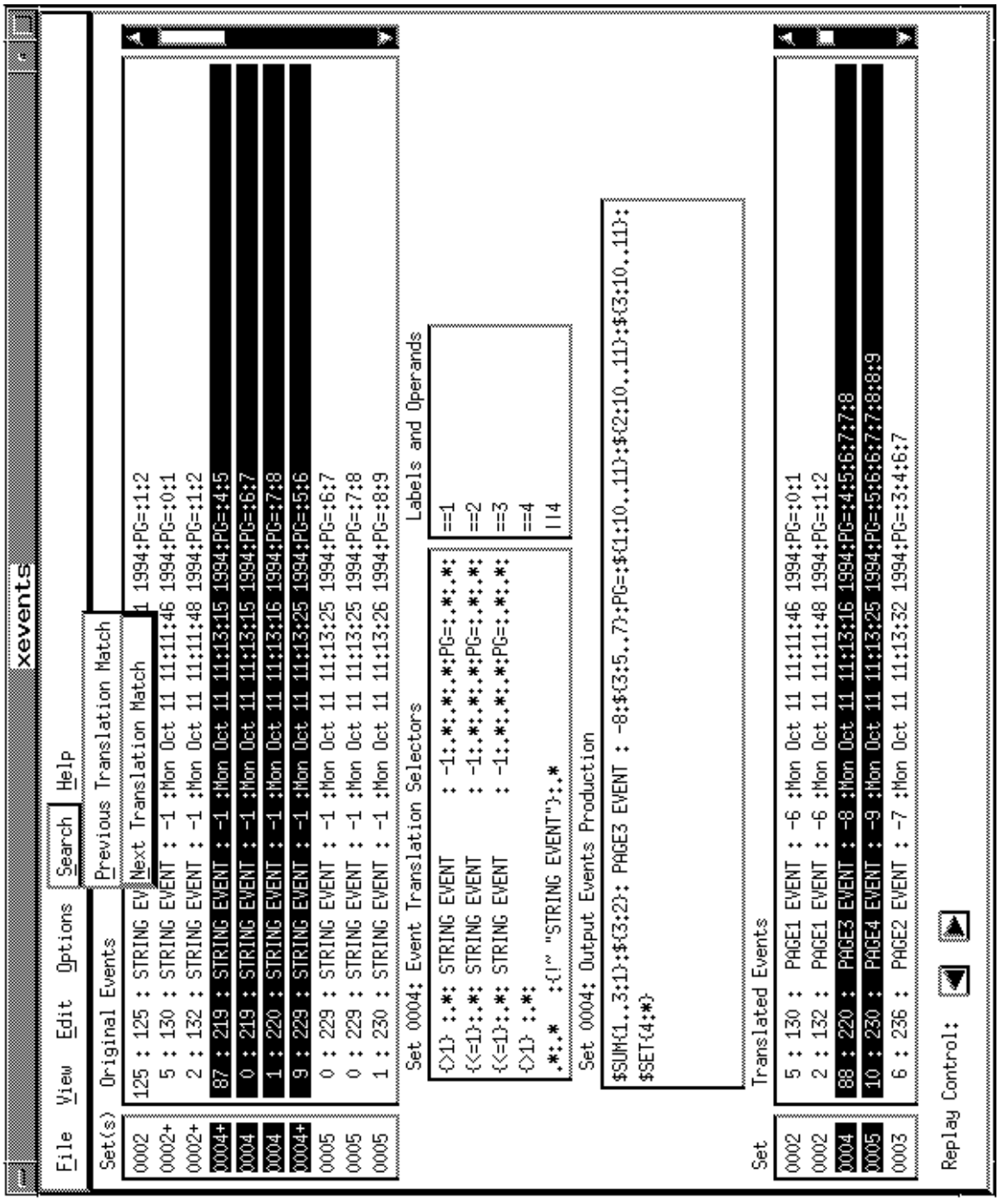


Figure 4.6 Example **Xbro***wser* display from a prototype of **xevents**

A recent addition to the **Xbro***wser* system is **xevents**, a prototype of an event record replay tool that can also be used to interactively filter the event output from **catevents**. Fig. 4.6 is a example of the proposed interface of **xevents** when editing event filters. When replaying an event record a small movable **xevents** window (not shown) appears in front of the full set of **xbro** displays that produced the record: as the vcr-like controls are used, the appropriate **xbro** displays are created, simulating button-pushing and editor-filling either in a time-based mode with variable replay speed or in an event-based mode that allows single-stepping. The editing view shown in the figure has a menu bar at the top. The uppermost windows contain a scrolling list of the original event record. Just below, a collection of windows show for the current event translation set the event translation selector masks with associated line-oriented labels and operands, and below that the output filter control window. At the bottom is a pair of windows for the resulting translated event record. The input event record and output event record windows display matching highlighted lines to indicate lines selected for event translation and the corresponding output events produced; "Set" indicators show the set identifiers of event translation rules that match original events or produce the translated events. The current prototype of **xevents** stores translation sets as awk scripts that may be altered manually as needed.

4.4 Experiences using Xbro*wser* with an extensive study

The study reported in Chapter 7 used **xbro** documents that were shared by all subjects as well as some uniquely produced for each subject. Customized dvi-files contained 24 random problems covering all combinations of experimental variables arranged as 12 sets of two problems. As each problem set was used, the corresponding dvi-file was uncompressed by commands embedded in hypertext annotations in the previous problem, and the preceding set was compressed. Each problem set required 329 kilobytes of storage uncompressed, and 96 kilobytes of storage compressed for a maximum of 1.3 megabytes per subject while working problems. The total size of dvi-files shared by all subjects was 1.9 megabytes.

Event files stored for each subject covered an average three and half hours of participation involving almost 800 displays and filling in 300 editors, and when complete were typically less than 64 kilobytes uncompressed and 20 kilobytes compressed. The corresponding final context files for each subject averaged 40 kilobytes uncompressed

and less than 10 kilobytes compressed. Event and context files are optionally stored in compressed form between uses, but are not usually an issue while an experiment is being administered.

Display speed was carefully considered: a complicated full-screen set of displays such as in Fig. 4.3 required less than one second to complete. More commonly, only one-fourth of the screen was updated at a time, in under one-sixth of a second; such updates can be done without clearing the entire window if only button, indicator, or editor status changes, further reducing distraction owing to display refreshing. The implication of these maximal display times is that experimenters using **Xbro***wser* can expect to detect effects on the order of seconds, as might be expected for an event-level protocol recording system. In this experiment, the shortest time interval with significance in an analysis of variance on was about 10 seconds; however, this had $p < 0.001$ with 18 subjects and 216 observations, implying that weaker effects (*i. e.*, smaller intervals) should be readily detectable.

For data analysis **catevents** and awk were used extensively: to summarize time spent in each section of the experiment and time to complete each problem, to generate a trace of all errors made by subjects and to score those error traces in terms of severity, and to produce data files in a format that could be read as input to SAS[13] procedures used for statistical analysis. Several versions of tracing and scoring awk scripts were used with several styles of SAS input files to verify the output interface of **catevents**.

The prototype of **xevents** was developed using an awk script that converts earlier **Xbro***wser* event records into the latest version; event records contain history fields describing the versions of **Xbro***wser* and **xevents** used to produce and edit them. To maintain the integrity of data, alterations may be stored as edit operations appended to a copy of the original data, or as a final edited version to save space.

4.5 Future directions for Xbro*wser*

The experiment described in Chapter 7 shows how valuable the current version of **Xbro***wser* can be in collecting and processing extensive amounts of data. Three major classes of improvement are currently envisioned, covering authoring, implementation, and future plans.

[13]SAS is a trademark of SAS Institute Inc.

First, the LaTeX macros currently provided for hypertext document authoring are to be augmented by a set created explicitly for the aforementioned study, including automated multiple-choice buttons, visibility-toggling indicators, and highlighting completion indicators. Source for other complex macros from that study will be distributed with **Xbro***wser* as examples of ways to produce various effects.

Second, several implementation issues of **xbro** could be improved: display and hypertext interface code could benefit from improved speed, the editor interface should be upgraded to use the more flexible Motif or XView user interface standards, and the event record format should include basic events previously implemented as extended string-event markers, such as host and display machine usage and display/page activity.

Finally, planned extensions to the **Xbro***wser* system remain to be completed, such as a full implementation of **xevents**, focusing initially on interactive replay of event records with the corresponding hypertext documents. Another feature of **xevents** to be completed is its ability to interactively build awk or perl scripts to filter event records to produce desired output time-based, session variable or editor content-based values in a flexible fashion.

4.6 Conclusions

The **Xbro***wser* system provides a valuable tool for empirical tests of user interface issues. By using high-quality page layout programs as a basis, many opportunities arise including rapid prototyping of user interface layouts simulating program appearance, menu and dialog design, and preparation of text manuals. The common dvi-file substrate makes it trivial to conduct text-*vs.*-display experiments guaranteeing the only difference between materials is the respective quality of the text and computer display output devices and the method used to turn pages. **Xbro***wser* also provides a platform that can provide high-quality hypertext manuals and instructional materials. With its protocol analysis capabilities, **Xbro***wser* makes it possible to design, implement, and analyze complex user interface studies efficiently and unobtrusively.

Part II

An Empirical Study of Parallel Program Comprehension

This section of the dissertation discusses an extensive empirical study of dependence analysis using the **Xbro***wser* system. A version of the data similar to that used for statistical analyses reported here will be made freely available over the Internet. For more information, contact the author by email at `bro@cs.rice.edu`, or consult the World Wide Web page: `http://www.cs.rice.edu/~bro`.

Chapter 5

Empirical Studies of Programming

The importance of understanding human abilities and limitations in the area of parallel programming is clear. One approach to achieving that understanding lies in using empirical studies to measure comprehension of dependences and data-flow in sequential and parallel programs.

There is a rich literature on methodology for empirical studies of programmers, but virtually no application has been made to parallel programming [Wad93]. Empirical studies of sequential programming provide details of experimental design useful in the implementation of the study described in Chapter 7. Pertinent reviews of the literature that provide design methodologies include Shneiderman [Shn80], Brooks [Bro80], Sheil [She81], and Landauer [Lan88]. Broader works in the more general context of human-computer interaction can be found in Curtis' description of the paradigms of psychology of programming [Cur88], Boehm-Davis' survey of program comprehension studies [BD88], and Egan's discussion of individual variance [Ega88]. Another useful survey of studies of programmer variance appears in a review of statistical methodology in the literature on human capabilities by John Hammer, who addressed the question of statistical power in computer programming, concluding that power is not generally a problem with such work [Ham84].

In addition to these survey and methodological works, two areas of specific studies have proved helpful to the research described in this dissertation. These include studies of the use of control- and data-flow in sequential programming analogous to the current research, and studies of individual programmer variance. In the former category, Weiser's work with slicing [Wei79] and Lyle's work with dicing [Lyl84] have been particularly helpful, while Iselin's experimental design [Ise88] is most similar to the current effort in that he collected both error severity and time to demonstrate comprehension; in the latter category, Oman, Cook, and Nanja's experiment comparing the effects of programming experience on the debugging of semantic errors [OCN89] is most similar, again for its use of both error severity and time to task completion as dependent measures in their study of interactions of programming experience with

the nature of programming problems. These and other related works are discussed in some detail here.

5.1 Control- and Data-flow Use in Programming Studies

Although involving control-flow as well as data-flow, Weiser's and Lyle's works on slicing and dicing are best attributed to the data-flow aspect of these studies, while other research more directly addresses the effects of control-flow inherent in program loop and conditional statements.

5.1.1 Data-flow Work: Slicing and Dicing

Weiser's Slicing Experiment

Mark Weiser used empirical techniques in his dissertation on the data-flow technique known as *slicing* [Wei79]. Slicing involves using traditional data-flow techniques to eliminate program statements that do not contribute to the values stored in a particular set of variables at a particular place in a program. The statements that remain are called a *slice* of that program for those variables at that program statement. In later work he described a test of whether programmers mentally construct slices when debugging [Wei82]. Weiser's hypothesis was that while debugging, programmers use data-flow techniques and program slices as mental representations of program specifics. Such knowledge is not conceptualized as linear lists of program statements but as connected threads related by data-flow.

Three ALGOL-W programs were prepared, each with a simple bug. In addition representative code fragments from each program were selected, consisting of slices, contiguous chunks, or random statements, some of each including the bug statements. Each subject was given a randomly selected program to debug. The program description and output could be referred to at any time. After finding a correction, the subject recorded the time and went on to another program. After debugging all three programs, a short break was given. The subject then was shown code fragments and asked to rate how confident he felt that the code had appeared in one of the three programs.

When analyzed, the difference in ability to recognize relevant and irrelevant slices was significant, as was the difference between relevant slices and random statements. Slices relevant to the planted bugs were recognized more often than slices not con-

nected to the bugs, and also were recognized more often than collections of random statements from the programs.

The conclusion was that programmers used slices internally to encode knowledge about programs being debugged. However, there are classes of bugs that cannot be found using slicing [WL86]. Human reasoning based on innate slicing techniques will fail on those classes of bugs. It remains to be determined whether similar innate data dependence techniques and flawed reasoning based on them exist.

Lyle's Slicing Experiment

Jim Lyle's dissertation has a description of another slicing experiment [Lyl84]. The initial hypothesis was that an experimental tool called **Focus** that included a slicing aid would improve performance on debugging tasks. **Focus** was implemented as an experimental test-bed for evaluation of slicing-based tools.

For the slicing experiment, two FORTRAN programs were produced, each with randomly introduced bugs. All subjects debugged both programs with versions of Focus, with one program selected at random used with the version with a slicing tool added, while the other program would be used with the version without it.

Debugging times with and without the slicing tool were compared for the two programs. There were no statistically significant differences. Despite the lack of conclusive results, the slicing experiment provided some interesting information derived from post-experiment interviews. Most subjects found the tool useful. Negative comments were directed at the user interface rather than the tool itself. This interview technique can recover data that might be otherwise overlooked.

Another useful technique from the slicing experiment was the use of a small number of subjects in a pilot study. This allowed small problems in experimental design and materials to be corrected before time and expense were invested in a large study; this approach was used extensively in the current study.

Lyle's Dicing Experiment

The major contribution of Lyle's dissertation was the introduction of a technique called *dicing* based on slicing [Lyl84]. A *dice* is constructed from a slice of a program by removing statements that appear in slices on variables known to be correct. The remaining statements must contain the bug if the original slice did.

This follow-up to the slicing experiment simulated on paper the effects of using a dicing tool. Lyle expected dicing to improve programmer debugging speed. A FORTRAN program was used, with three planted bugs. The practice treatment for

the experimental group explained what dicing was, and how it could help in debugging programs. Both groups were shown the same practice program, with the experimental group also being shown sample slices from it. This was followed by an explanation of the target program both groups were to debug. The control group was provided a listing of the entire program, while the experimental group was also given slices for each error with the statements in the dice marked.

The control and experimental groups were statistically analyzed to ensure their background and experience were not significantly different. This data was collected with a background questionnaire. Lyle found that there was no significant difference in the biographical histories of the control and experimental groups. Further, he found that the dicing group was significantly faster at debugging for two of the bugs, and on the border of being significantly faster for the third.

Uses of Slicing and Dicing in Debuggers

One variant of slicing described by Korel and Laski [Las90, KL90] is *dynamic slicing*, in which a specific set of inputs to a program is considered in the process of constructing relevant slices by dropping code dependent on control-flow that cannot occur for that input. The original (or *static*) form of slicing assumes all control-flow paths may occur, and presumes no knowledge of content of variables. By taking advantage of the additional information and by operating on a program trajectory (list of executed statements) rather than the original static program, a dynamic slice may be much smaller than a static slice, and so more useful for debugging purposes. Another significant extension of dynamic slicing is its ability to deal with pointers and records. Dynamic slicing was used in the implementation of the debugger known as STAD (System for Testing And Debugging).

Another tool that uses dynamic slicing as well as dicing to reduce slice size is C-debug, described by Samadzadeh and Wichaipanitch [SW93]. A debugging tool for C language programs [Sch90], C-debug was designed to operate as a utility program running under the UNIX[1] operating system. In the form presented, it did not yet support structures, unions, or defined-types, but it did handle control structures, function calls, and pointer expressions.

[1] UNIX is a trademark of AT&T.

5.1.2 Control-flow Work: Loops, More Loops, and Learning

Cognitive Strategies and Loops

Soloway, Bonar, and Ehrlich [SBE83] describe an experiment that compared error performance using a programming language designed to support specific loop strategies. They identified two strategies commonly used: "read/process" and "process/read". In the former, an unconditional loop reads data, tests for completion, and if not, then processes it. In the latter, an initial read precedes a loop that conditionally executes with a test for completion, embodying processing of the data read and reading the next data. Soloway *et al.* believed the process/read approach, commonly achieved with constructs such as the "**while**" loop of Pascal, to be more awkward than read/process designs, that may be implemented directly with Ada's "**loop... leave... again**" construct. They hypothesized that people would find it easier to program correctly with a language that directly supported their preferred strategy, and tested this with a two part experiment. First, to determine a preferred strategy, they asked subjects to produce a high-level "plan" for solving a programming problem. Second, they asked half the subjects to implement their plan using ordinary Pascal, and half with Pascal-L, a version in which the only loop construct was "**loop... leave... again**". They found that the preferred plan was to use read/process loops, but that subjects tended to implement a strategy matching the language support; subjects were also more often correct when using the language-matching strategy. Correctness, ability to sense the strategy matching the language, and preference for a strategy all seemed to improve with experience.

Cognitive Strategies and Loops, Revisited

Iselin [Ise88] studied some of the issues raised by Soloway *et al.* in a larger study that also addressed conditional statements and learning effects. Iselin using a program tracing task similar to hand-simulation to control confounding variables better [Pen82] with the language COBOL, measuring the length of time required to work out what output would be produced, the number of errors in the output, and the magnitude of the error in the output variable. Subjects were advised that time taken and errors produced were being measured and that subjects were not working the same problems. The results confirmed that experienced programmers did better that students, and that performance improved with learning. However, read/process loops were only better for student programmers: there was not even a trace of the expected effect for professional programmers. This could be an artifact of Iselin's use of COBOL, but

he also found a similar effect for positive *vs.* negative conditional expressions among the students that did not extend to the professionals.

Learning About Control-Flow

Kessler and Anderson [KA86] conducted two studies that dealt with issues involving learning about iteration and recursion and transfer effects between the two. Problems were designed to be recursive or iterative, operating on lists or numbers, accumulating results forward or backward in either failure or success conditions, and to either skip incorrect occurrences or to eject (terminate) on failure to accumulate; only the first and the last issues had a measurable effect.

The language used in the first experiment, SIMPLE, uses English-like syntax and LISP-like semantics, and was designed to study the acquisition of recursive programming skills. Subjects – essentially novices to programming – were randomly assigned to learn iteration or recursion first, and were required to solve a problem correctly before moving on to the next. It took between 2 to 5 hours to complete the entire experiment. Time to criterion and number of errors made were both recorded. The results showed that the group that learned iteration before recursion did better in terms of time on the second condition than did the recursion-iteration group; of the within-subjects conditions, the "on failure to accumulate, skip occurrence" problem took longer to get correct.

A second experiment replicated the first, using a version of SIMPLE with a simplified loop construct and a slightly different experimental design. In this version, it took between 2 to 5 hours to complete the experiment. The results confirmed that the iteration-first group did better on its second trial than did the recursion-first group, and changing the skip/eject factor was more difficult to deal with than changing the success/failure factor.

5.1.3 Implications of Control- and Data-Flow Studies

Weiser's and Lyle's works were instrumental in pointing out the value of looking at how programmers deal with data-flow issues in programs, and how the results can have concrete implications for tools and techniques generally useful in programming; the manifold uses to which slicing has been put are particularly impressive.

The studies on loop-usage and iteration reveal some of the deep interactions of language issues with suspected cognitive factors that affect performance; in addition, experimental design issues involving subject preparation in terms of expectations,

nature of data collection, and the importance of training are involved. Iselin in particular discusses some of these issues in some detail in presenting his experimental design, as well as collecting both time and error data, analogous to the study presented in Chapter 7.

5.2 Studies of Programmer Variance

Lyle's dicing study provides a useful example of motivations and techniques for controlling for programmer variance. Variance between experimental groups can generate statistical noise and obscure results. In Lyle's biographical questionnaire, subjects answered questions about length of programming experience, number of classes in computer science, number of programming languages used, and how comfortable they felt using FORTRAN. The factors used by Lyle are also among those identified by Hammer as useful in controlling programmer variance [Ham84].

5.2.1 Hammer's Survey: Controlling for Programmer Variance

Hammer discussed such factors in a larger context, using techniques to evaluate the statistical quality of the conclusions drawn regarding the empirical designs used in the studies he cited. He found that empirical studies of this type demonstrated adequate statistical power to warrant the conclusions they reached.

For professional programmers, years of programming experience were a fair predictor for program reading and writing performance. Experience and number of computer science courses correlated well with program writing time. For professionals with less than 3 years of experience, the number of known programming languages and the number of familiar FORTRAN programming concepts correlated with debugging performance.

For advanced computer science students, several promising predictive measures were found. Multiple correlations exist between program comprehension and writing tasks and the number of computer science courses taken, grade point average in those courses, and years of programming experience. These three factors also tended to be mutually independent.

For beginning formal programming students, previous years of programming experience, SAT[2] Mathematics scores, or college course grade(s) in introductory program-

[2]Scholastic Aptitude Tests

ming, calculus, or chemistry were useful data. Recording experience with personal computers was also recommended.

5.2.2 Specific Studies in Programmer Variance

The following paragraphs contain further specific examples of research that studied background data, frequently with error- or time-based measures of performance, to account for individual variance.

Performance in Computer Programming Courses

A specific example of this approach can be found in the work of Koubek, LeBold, and Salvendy who correlated the academic achievement in high school and college of more than 3000 students to their performance in computer programming courses [KLS85]. Koubek *et al.* also describe related work that found significant correlations of programming performance with overall grade point average and SAT Verbal and Mathematics scores. In their own research, they found that performance in high school and college mathematics and science courses accounted for up to 50 percent of individual variance; specifically, they found that high-school mathematics and science course grades, along with SAT mathematics scores were the best pre-college predictors of performance in introductory programming courses, independent of the programming language taught or the student's major course of study. By supplementing these with college science and mathematics course grades, the regression equations improve still more, and are able to account for 25 to 50 percent of the variance in performance. These background factors also significantly correlated with performance in advanced programming courses.

Benefits of Collaborative Effort

Wilson, Hoskin, and Nosek tested the benefits of collaboration for student programmers in a study that compared performance on problem-solving involving writing computer programs for individuals and two-person teams [WHN93]. They found evidence that performance in terms of readability of the solutions was improved by teamwork, and that the most measurable impact of collaboration was in heightened post-experiment measurements of confidence in the solution and enjoyment of working the problem. These latter results show the value of measuring such "soft" indices when considering performance in inherently reductionist experimentation: in the absence of performance contraindications, improvements in such indices may validate the existence of longer-lasting effects, such as reduced fatigue or improved job-satisfaction.

Student Populations Differences in Motivation

A similar "soft" consideration in background studies involves issues such as motivation. Wilson and Braun presented a study that related choice in different academic programs to differences in motivation [WB85]. They found that, compared to computer science students, business information systems students thought computer professionals should be more practical, could legitimately be more interested in making money through their work, and did not necessarily have a high mathematical aptitude; they also were less likely to consider it reasonable to sacrifice documentation in the interest of storage efficiency, and were less trusting of computers in general. Their results also point out possible effects of motivation on general performance, suggesting that testing for motivation or other related background differences is a good idea where populations being studied may include different groups.

Debugging Semantic Errors

Oman, Curtis, and Nanja provide some evidence of the effect of programming experience on debugging error [OCN89]. They compared performance of novice, intermediate, and skilled computer science students trying to debug a binary-search and a median-calculation program. Each program contained a single semantic error involving either violation of array bounds or use of an undefined variable, and was presented to subjects with an appropriate error message that either did or did not include a line number indicating where the error occurred. Subjects were requested to circle the incorrect statement, to write a corrected version, and to record the time they completed the debugging process. Errors were scored in terms of severity on a scale from 0 to 3: 0 = bug not located, 1 = bug found but not corrected, 2 = bug found but correction syntactically incorrect, and 3 = bug found and correction completed. The results support Gould and Drongowski's conjecture that if a programmer can find an error, it will be corrected successfully [GD74]: of 386 debugging trials, there were only seven instances where the bug was located and some form of corrected statement was not provided. Experienced programmers were most successful in locating and correcting the errors, and did so faster than less experienced programmers. In addition, they were less dependent on debugging clues, such as the line numbers experimentally provided. For intermediate programmers, inclusion of the line numbers improved performance but was still less than that of experienced programmers, while novice programmers were mostly dependent on the line number clue. This particular study shows the value of including programming experience as

a background datum of interest, as well as providing an example of an experiment that records both error severity and time, as does the one I describe in Chapter 7.

Productivity of Programming Teams

A final example concerns the experimental design of Gowda and Saxton for a study of factors influencing the productivity of programming teams [GS89]. They took advantage of the opportunity provided by an ACM[3] regional programming contest held in Ohio to study the influence of individual and group characteristics on programmer productivity. To this end, they collected questionnaire data from fifty-seven four-person teams including questions on graduate/undergraduate status, SAT and GRE[4] scores, grade point average, list of courses useful in preparing for the contest, and programming experience. Their preliminary results focused on team and leadership style, and indicated that the most successful teams divided labor much more quickly and worked in coordinated two-person problem-focussed sub-teams.

5.2.3 Implications in Questionnaire Design

The absence of factors relevant uniquely to parallel programming is the result of the absence of empirical studies directed at parallel programming problems. Based on these sequential programming studies, I generated an extensive background questionnaire about topics likely to be relevant to measuring variance in parallel programmers, including years of parallel programming experience, percentage of programming time spent in parallel programming, number of courses directly addressing concurrent or parallel programming, and number of parallel programming languages or concepts known. Questions less certain to be useful examined experience in analogical situations, such as managing large numbers of people. The data from this questionnaire was used both as in Lyle's experiments as a verification of the comparability of experimental groups, and as suggested by Hammer as factors to test for correlation with success or failure in parallel programming; both uses are reported in Chapter 7.

[3] Association for Computing Machinery

[4] Graduate Record Examination

Chapter 6

Parallel Programming and Dependence Analysis

Parallel programming is a moderately recent programming paradigm motivated by the ability of parallel computers to improve execution speed at a fraction of the cost of increasing the speed of a conventional sequential computer. Unfortunately there are known data-flow difficulties called data dependences, described in Section 6.2.1, that prevent some programs from being readily converted to parallel forms. One technique for identifying and correcting such problems is dependence analysis, using knowledge of the nature of dependences to detect and correct them where possible. My research uses dependence analysis in two ways. First, the potential value of classical dependence analysis is one factor to be empirically studied. Second, dependence analysis techniques are extensively used in the generation of the programming problems presented to subjects.

6.1 Parallel Programming

Parallel programming is the process of specifying a program designed to execute on a parallel computer. Just as a conventional sequential computer executes one instruction at a time, a parallel computer is capable of executing several instructions simultaneously. The speedup possible using a parallel computer is thus proportional to the number of instructions it can execute at one time. To improve a sequential computer, basic technology must be improved at potentially exponential cost. Parallel computers on the other hand may be constructed by combining existing sequential computers to work in parallel, at a linear cost. Fig. 6.1 gives an example of this type of cost-benefit analysis. Improving sequential computers may require fundamental changes in design or technology. Improving parallel computers may only require buying more processors; indeed, improvements in sequential computers can be incorporated in parallel computers simply by taking advantage of new technology as it becomes inexpensive.

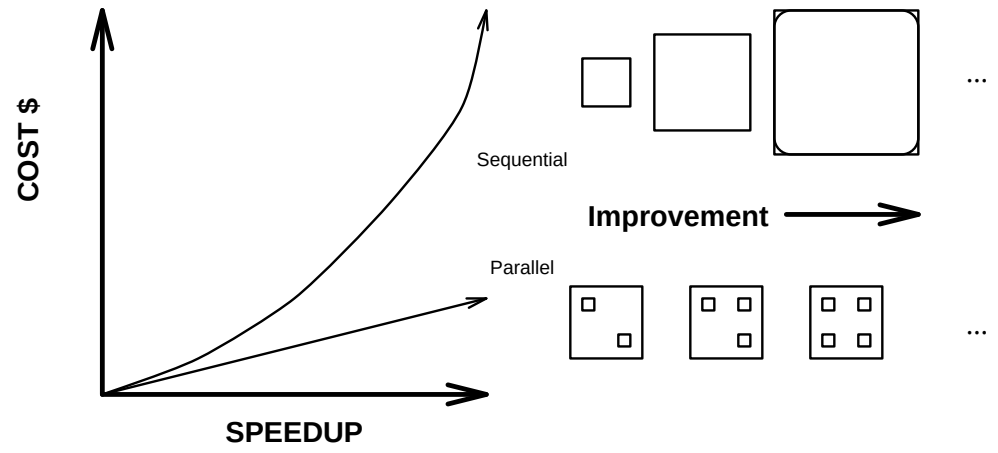


Figure 6.1 Cost-speedup comparison for sequential and parallel computers

One obstacle regarding parallel programming concerns data-flow problems that prevent parts of programs from executing in parallel, while a program can always be improved by executing its component statements faster. Consider the statement "`a = 1`" followed by "`a = 2`". They may be executed at any sequential speed desired as long as their canonical order is preserved, and the final value for `a` will be 2, but if executed in parallel the first statement could complete after the second and produce the final value 1. This *dependence* on execution order between data references in a program's statements is the central problem of parallel programming that dependence analysis addresses.

6.2 Data Dependences and Dependence Analysis

6.2.1 Data Dependences

Kuck, Kuhn, Padua and Wolfe identify basic dependence relations between statements in an algorithm or program [KKP+81]; Bernstein provides a succinct summary of the underlying data-flow conditions [Ber66]. These have been determined to be critical to the understanding and construction of parallel programs [AK87, SLY89, Hag90] and are equally important in sequential programming transformations that affect execution sequence, such as reversing a loop. Dependences are of two classes. *Control* dependences result from the control structure of the algorithm. *Data* dependences are concerned with the flow of values in the execution of the algorithm, and imply a

required ordering to preserve program semantics. Changing the order of statements involved in a data dependence *violates* or *breaks* the dependence. There are three basic types of data dependences:

- An *output* dependence occurs when two statements define the contents of the same memory location. The output of the program will depend directly on which statement executes last.
- A *flow* dependence (or *true* dependence) occurs when a memory location is set by a statement and is read by another statement later in a determined sequential ordering.
- *Anti-* dependences occur between statements in which one uses the contents of a memory location that is set by another statement later in a determined sequential ordering.

In addition to these simple data dependences, the presence of control structures may affect them. Examples of this include *loop-carried* dependences, in which one of the basic types of data dependence occurs between instances of iterations of a loop body. Thus each of the previous simple data dependences may also appear in a loop-carried form; the alternative is known as *non-loop-carried* or *loop-independent.* Loop-carried dependences of concern typically involve array references rather than scalar variables.

6.2.2 Uses of Dependence Analysis

Why are dependences of interest in the empirical study of parallel programming? Several observations provide the answer.

Allen and Kennedy noted that the concept of dependence as used in an ordinary data-flow analysis is substantially different than in parallel programming research [AK87]. In most sequential data-flow analyses, dependence is used to imply which statements must be present for others to receive the correct values. Anti- dependence and output dependence are not usually needed for those purposes, serving only to determine the relative order of statements, a function crucial in parallelization. However, certain classes of loop transformations including reversal and parallelization require awareness of all three types of dependences

Shen, Li, and Yew determined in an empirical study of array-based dependences in FORTRAN programs that flow dependences – used in both sequential and parallel

data-flow analyses – are most common, producing 63% of the dependences encountered, with anti-dependences making up 3%, and output-dependences 34% [SLY89]. They insist that the state of the art in data dependence analysis and automatic parallel execution techniques was unable to completely fulfill the needs of parallelization, relying on *user assertions* to improve performance.

Haghighat, whose work extended the ability of dependence analysis to handle some of the problems identified by Shen *et al.*, also noted the necessity for user assertions to supplement automatic techniques [Hag90].

Callahan, Cooper, Hood, Kennedy and Torczon further supported the need for user interaction, and described uses of dependence analysis techniques for programmer assistance in PTOOL and in its successor, the ParaScope programming environment [CCH+88a, CCH+88b].

The conclusion is that direct human involvement in the dependence analysis process is necessary to guide automatic algorithms, while the lack of empirical studies to measure human abilities in using dependence techniques suggests a significant research area that must be investigated. A handful of statements can generate a dense tangle of dependences, any one of which can prevent parallelization. Dependence analysis has become a major technique in the study of parallel programs, and is used as an analytic tool in parallel programming environments, but difficulties humans may have in applying it effectively remain unknown. The current study attempts to open this hitherto closed door with the assistance of empirical techniques using **Xbro***wser*, a program designed to present hypertext experimental materials and collect data for protocol analysis.

Chapter 7

An Empirical Evaluation of Dependence Analysis in Parallel Program Comprehension

The analysis of data dependences resulting from data-flow references to memory locations in the execution of a program are critical to understanding and constructing parallel programs. [KKP+81, AK87, SLY89, Hag90] My intent in this study is to investigate several factors related to dependence analysis that potentially affect program comprehension, applying empirical methods commonly used in the study of sequential programming to measure effects on error severity and time required to show successful comprehension of simple loop-based program transformations related to parallel programming. In addition, I investigate correlations of subject background factors to these same error and time values controlling for significant experimental results.

The results show that graphical annotation of program source as is done in the ParaScope parallel programming environment improves the time required to correctly comprehend the results of parallelizing loops, and that dependence type affects both time required for successful comprehension and severity of errors made in the attempt.

The investigation of background factors revealed that, controlling for the experimental effects, several factors related to mathematics ability and education such as standardized test scores, number of courses taken, and grade point average correlated to improved error and time performance.

These results also demonstrate the efficacy of the **Xbro***wser* hypertext system for protocol analysis described in Chapter 4, used to administer the experimental materials, including training and testing of subjects, collection of data in an interactive questionnaire, and event-level subject protocol recording.

7.1 Related Work

Gowda and Saxton [GS89] describe the administration of an experiment in which observations were made of a programming contest in which the number of problems

solved and time to complete problems were recorded. Videotape and audiotape were used in some cases, and all subjects completed pre- and post-contest questionnaires. Later work in the same line by Gowda and Chand [GC93] demonstrates the difficulties in measuring results on programmer productivity derived from background factors: dependent measures can be swamped by statistical noise if the scale of measurement is too broad. Difficulties such as these point up the value of survey works such as Boehm-Davis' of software comprehension [BD88], making it possible to objectively consider possible dependent and independent variables in light of previous experience.

Sometimes this itself is difficult: virtually no empirical work has been done on parallel programming [Wad93], making the task of experimental design harder. Still, sequential programming studies can provide valuable examples. Oman, Curtis, and Nanja [OCN89] describe an experiment in which time to debug and debugging error severity were measured for subjects with novice, intermediate, and expert programming backgrounds on several problems dealing with program semantic errors including array bounds and undefined variables: experts were better at finding and correcting bugs, and needed less support to do so, while intermediate subjects were better but needed more support, and novices suffered without the most extensive support. Soloway, Bonar, and Ehrlich [SBE83] describe an experiment that compared error performance using programming languages designed to support specific loop strategies; they found a natural preference for "read/process" loop strategies, that subjects did better with a language supporting that strategy with a "**loop... leave... again**" construct, and that experience reduced subjects' susceptibility to these influences. Iselin [Ise88] replicated these results in a study that also included effects of conditional statements and learning: again, experienced programmers did better that students, and learning improved performance, but the benefits of the read/process strategy could only be confirmed for programming students, not professional programmers. Kessler and Anderson [KA86] have studied the process of learning flow of control, finding that such measures do reflect underlying difficulties for subjects, and that learning iteration transfers to learning recursion better than the reverse condition. The work most similar to this research is that of Weiser [Wei82, Wei84] and of Lyle [Lyl84] [WL86], who describe extensive evidence that programmers are influenced by control- and data-flow considerations in the context of *slicing* (selecting only program statements relevant to control- and data-flow producing a variable's contents at a given program location) and *dicing* (selecting program statements from a slice on an incorrect variable by removing statements known to produce correct variables).

Weiser found evidence that programmers use slices in mental models of programs, while Lyle extended the slicing model and provides proof supporting the efficacy of using both slicing and dicing in computer-supported programming environments. Later uses of slicing and dicing include that Korel and Laski [Las90, KL90], who describe work related to dynamic slicing in STAD (System for Testing And Debugging), and Samadzadeh and Wichaipanitch [SW93], who relate experiences using slicing and dicing concepts in the design of C-Debug, an interactive debugging tool.

The shortage of empirical data on parallel program comprehension also applies to studies of individual differences in parallel programming, requiring me to look to sequential programming studies for relevant work. Egan's survey of individual differences studies [Ega88] is a good introduction to some of the issues involved, and also discusses automated "mastery learning" techniques that tend to improve mean performance and reduce variability in achievement. These techniques, including modular lessons, hierarchic skill acquisition, and achievement testing coupled with remedial instruction, were employed in the instructional phases of the current study.

In another survey work on statistical methodology in human factors in computer programming, Hammer [Ham84] identifies several variables that have been used in sequential programming studies of individual differences: these contributed greatly to the design of an online interactive questionnaire used in the current work. Other examples include the work of Koubek, LeBold, and Salvendy [KLS85] who found that up to 50 percent of individual variance in performance on computer programming courses could be accounted for by performance in high school and college mathematics and science courses, Gowda and Saxton's consideration of SAT, GRE, grade point average, and programming experience in his experimental design [GS89], as well as Wilson and Braun's study of student background that found motivational differences in student populations that would appear to affect performance [WB85]. Wilson in later work with Hoskin and Nosek [WHN93] used grades and post-treatment satisfaction indices to show enhanced confidence in solutions and enjoyment of problems in a study of the positive aspects of collaboration for programming.

In this study, my goal is to investigate the effects of dependence analysis with the intent of applying the findings to present and future tools supporting parallel programming. Pilot studies described later were used to identify the five experimental factors described in the following section as being of immediate interest.

7.2 Empirical Variables

Dependences used here come in three types and may be thought of as an elaboration of Bernstein's Conditions describing circumstances in which memory references from two different program statements may proceed without regard to the order in which those statements are executed [Ber66]. In Bernstein's formulation, the canonical ordering of a read of a memory location followed by a write to the same location, a write followed by a read, or a write followed by another write to the same location all must be preserved to guarantee the semantic result of execution. In dependence analysis as described by Kuck, Kuhn, Padua, Leasure, and Wolfe [KKP+81] the first form (a read followed by a write) is called an *anti-* dependence, the second (a write followed by a read) is called a *flow* (or true) dependence, and the last (a write followed by a write) is called an *output* dependence, with the first statement in the canonical ordering called the "source" and the second called the "sink" of the dependence. These three dependence types, further detailed by Allen and Kennedy [AK87], form the levels of an experimental factor called DEPTYPE, with examples shown presently.

Any program transformation violating Bernstein's Conditions or equivalently a data dependence by changing the canonical ordering of the statements involved can result in different values being produced. Two such troublesome transformations are to *reverse* the execution order of the statements, or to execute them in *parallel* which has at least the implied potential of reversal. These two transformations constitute the levels of REVPAR, a second experimental factor, and are only two cases from a large class of reordering transformations mentioned by Allen and Kennedy [AK87].

In this study, I used reversal or parallelization transformations on loop statements to measure resulting effects on comprehension. In a loop environment, an additional factor derived from dependence analysis is involved, based on whether data-flow involved in a dependence occurs within a given iteration of the loop body or between iterations of the loop body. The former, called a *non-loop-carried* (or loop-independent) dependence, in unaffected by transformation of the loop statement since the order of execution of statements within the loop body is unaffected. The latter, called a *loop-carried* dependence, is adversely affected by loop statement transformations since the canonical ordering of statements involved in different iterations of the loop body is changed. These two cases make up a third experimental factor called CARRIED. Note that the source of a non-loop-carried dependence precedes its sink in a loop body, while the canonical ordering of loop-carried dependence statements is

determined by the array indices involved and is independent of the order of the source and sink statements within the loop body. Fig. 7.1 provides examples of all three levels of DEPTYPE for each CARRIED treatment, including source-first and sink-first loop-carried versions. For this study, source-first and sink-first versions of loop-carried dependences were randomly assigned, limiting the comparison for CARRIED to data-flow distinctions only. Kuck *et al.* [KKP+81] describe both varieties of dependence, but terminology used here is adapted from that of Allen and Kennedy [AK87].

One method used to improve comprehension of parallel programs has been automatic analysis of programs to show dependences to programmers who must deal with them. One technique of displaying dependences is to *annotate* program source with graphical indications of potentially troublesome dependences. The alternative is to require *manual* identification of possible dependences to be resolved by program-

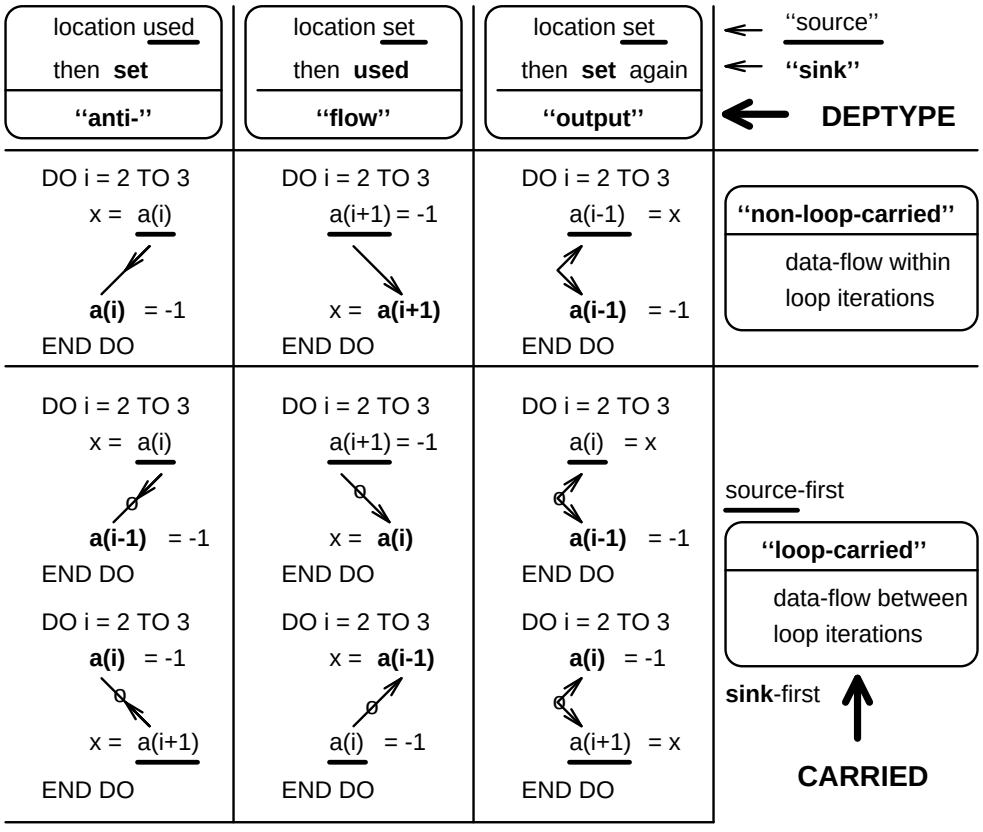


Figure 7.1 Examples of DEPTYPE and CARRIED treatments

mers before troublesome transformations will be successful. These two approaches are the levels of a fourth experimental factor called ANNOTATION, using annotation similar to that of Fig. 7.1. The annotation approach to assisting comprehension of parallel programs has been used in several parallel programming environments, including the ParaScope Editor mentioned in Cheng's survey of parallel programming tools [Che93] and described in more detail by Calahan, Cooper, Hood, Kennedy, and Torczon [CCH+88a, CCH+88b].

Finally, alternatives to dependence analysis can be considered in this empirical framework. One possibility is to use a simpler method, since anti- and flow dependences can be treated as special cases of a "read-write" memory access conflict (with output dependences a "write-write" class of conflict). Similarly, non-loop-carried dependences do not matter in loop transformation and so their emphasis may be reduced, providing less distraction from other concepts involving loop-carried dependences. Of course, identifying a dependence is not enough: array elements ultimately affected by it must be determined as well. In this study, a simple form of more general algebraic formulae were used for both TRAINING methods for that purpose, derived from extensive personal techniques developed to understand data-flow consequences of reversing program loops. This derivation is analogous to that of slicing by Weiser [Wei79] or dicing by Lyle [Lyl84] from observing programmer behavior. This alternative method is called "Algebraic Formulation", a name that reflects those formulae and was chosen for its suitability as an alternative to "Dependence Analysis" that together are the levels of the fifth experimental factor, TRAINING.

Given these five experimental factors, several hypotheses may be tested:

• **1** : Do differences in error severity or comprehension time exist for the Anti-, Flow, and Output dependence levels of DEPTYPE?

• **2** : Of the CARRIED treatments, are Loop-Carried dependence problems more error-prone or time-consuming than Non-Loop-Carried dependence problems?

• **3** : For ANNOTATION, does the Annotated method provide a benefit to programmers compared to a Manual method?

• **4** : Contrasting Reversed-loop to Parallelized-loop transformations for REVPAR allows comparison of similar sequential and parallel programming tasks: is parallel programming *inherently* more difficult when the task is similar to sequential programming?

• **5** : Finally, for TRAINING, is simplifying Dependence Analysis in the manner of Algebraic Formulation an improvement?

Additionally, interactions of these factors should be tested. Among those with a reasonable *a priori* expectation of detectable effects are CARRIED×DEPTYPE (if dependences matter only for loop-carried dependences), TRAINING×CARRIED (if glossing over non-loop-carried dependences matters), and TRAINING×DEPTYPE (if reducing the distinction between anti- and flow dependences matters).

In addition to testing these hypotheses, this study provided opportunities to try to relate subjects' background to performance on the measures used.

The effects of these independent variables were gauged on two types of response variables: an ERROR value that measures the severity of errors that occur when working on a programming comprehension task involving loop transformation, and a TASKTIME value that measures the length of time required to demonstrate adequate comprehension on that task.

These empirical factors were among those identified from the series of pilot studies described below as most immediately important.

7.3 Pilot Studies

Several pilot studies were used to develop hypotheses and materials. In the earliest version, three subjects read paper texts on parallel programming and answered written questions about the correctness of parallel-extended FORTRAN programs including both parallel DO and SELECT statements using P-F, a language derived for this research from FORTRAN 90 [BGA90]. The results suggested measurable results could be expected and that it would be advisable to concentrate on fundamental factors and a single programming language construct: the DO loop. The five factors used in the current study were chosen for specific investigation. One possibility seen in this first pilot study but set aside for later research concerns the advisability of creating parallel program constructs simply by adding a "PARALLEL" keyword to an existing sequential construct: some errors with such parallel constructs seem to result from confusion about parallel execution semantics influenced by the similar surface syntax. To lessen any such effect a matching keyword "SEQUENTIAL" was added to the P-F language used here.

A second pilot study was conducted using paper texts explicitly designed to study the five core experimental factors identified above. Four subjects were videotaped reading about parallel programming and working problems by filling in one-page checklists that identified a precise set of array elements affected by transforming DO

loops in a program. Colored flags were inserted in the text so that time values could be collected from the videotapes for each step of the process. A questionnaire derived from one used previously at a workshop on the use of the ParaScope programming environment was used to collect data on the background of subjects. Results of this pilot study suggested both time and error values were worth collecting, but that collecting time values unobtrusively on the scale required for the complete study would be difficult using written materials. Thus the **Xbro***wser* hypertext protocol analysis program described in Chapter 4 was designed, and the textual materials were adapted to use it for programmed-instruction and automated collection of data.

A final pilot study was performed with four subjects using **Xbro***wser* and hypertext versions of the materials used in the current study. The only major difference from the current protocol was the videotaping of subjects to help verify interaction with and data collection provided by **Xbro***wser*. The results confirmed the feasibility of the approach and the value of the data to be expected, and provided an estimate of time required to complete the experiment that was provided to subjects enrolling in the study.

7.4 Method

7.4.1 Subjects

Subjects used were volunteers from four advanced computer science courses taught at Rice University open to both upper-level undergraduate and graduate students. Volunteers received extra credit for participating. Subjects were randomly assigned to either the Algebraic Formulation or the Dependence Analysis TRAINING group when enrolling in the study through use of a shell script using random number generators to make the assignment, set the subject's chosen **Xbro***wser* username and password, install appropriate links to shared hypertext **Xbro***wser* files, randomly generate a set of 24 problems including answer keys covering all within-subjects experimental variables, and then produce customized hypertext **Xbro***wser* files for those problems using LaTeX [Lam94] and macros provided with **Xbro***wser*.

Because of hardware and software failures not related to experimental issues only 10 subjects in Dependence Analysis group could be used for data analysis, while 16 subjects in the Algebraic Formulation group were available.

After completing the study, an extensive background questionnaire was administered. A *post hoc* examination of this data revealed no major departures from the

assumption of TRAINING group background equivalence, and that subjects had an average of over eight years of programming experience, were extremely confident in their ability to do sequential programming, knew on average nine programming languages and had used an average of four within the last six months, spent roughly half of their working time programming, and had completed an average of eight college-level computer science courses. A complete report of the questionnaire results may be found in Appendix B.

7.4.2 Materials

A small computer laboratory was used for this study, containing six Sun 3/50 workstations configured as X-terminals, each with a 19 inch monochrome display of 1024×960 pixels with a 14 inch by 10.75 inch usable screen area, a keyboard, and an optical mouse that could be placed on either side of the keyboard. Instructions for using **Xbro***wser* were posted by the door. Users were not routinely observed while participating in the study other than through data collected by **Xbro***wser*, but an experimenter was available for consultation in another office, and could be reached by a telephone provided.

Xbro*wser* was used to present the hypertext programmed-instruction document summarized in Fig. 7.2, showing some of the hypertext links used for document navigation. Adjacent boxes were sequentially linked from top to bottom. In general, links branching forward were disabled until the destination was encountered in sequential reading. To simplify the diagram, reference shortcut links from the Index and within sections are not shown. Boxes with rounded borders were identical for both TRAINING groups except where passing references were made to group names, as indicated by "*" after the title. Dark rectangular boxes differed substantially for each group but were constrained to similar amounts and complexities of text. The number to the far right of each box is the number of individual window displays used to complete each section: thus the entire study involved showing 797 displays to a subject, not including optional ones such as the Index or individual review.

Identical Review problems were used for all subjects except for differences in TRAINING method. Regular problems worked by subjects were generated randomly by choosing problems from a set of templates covering all possible combinations of the experimental factors and choosing from a set of template variable names taken from twelve sample programs. This process produced FORTRAN 77 or C programs with

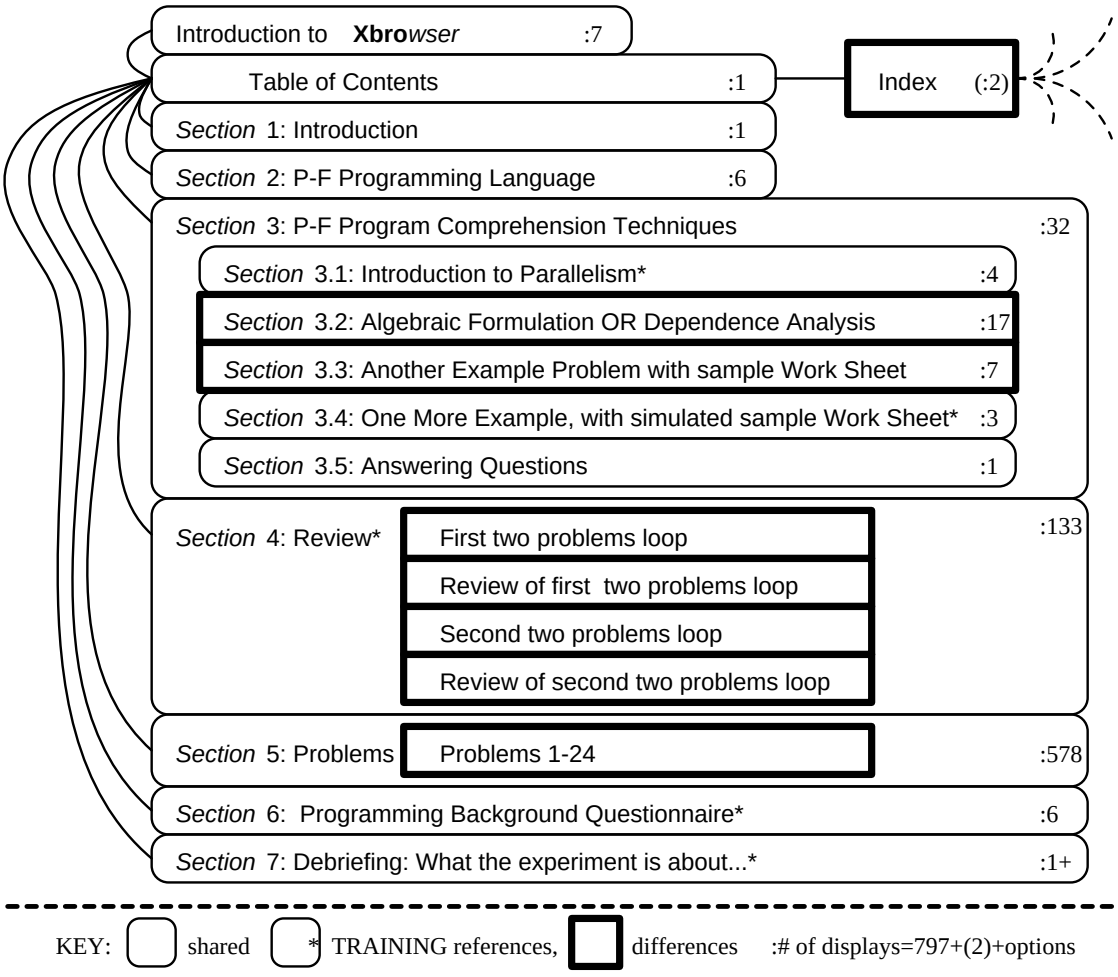


Figure 7.2 Overview of the **Xbro***wser* hypertext study document

forward and reversed versions of problem loops that automatically generated answer keys for each problem as well as customized LaTeX code to manufacture **Xbro***wser* files for each problem set.

A more complete description of materials appears in Appendix A.

7.4.3 Procedure

After an initial scripted demonstration in which they were advised that records would be kept of their specific interactions with the system, subjects used **Xbro***wser* to read a version of the document summarized in Fig. 7.2 appropriate to their assigned TRAINING group and containing randomly generated problem sets. Table 7.1 com-

Algebraic Formulation	Dependence Analysis
"violations"	"loop-carried dependences"
"wwrite" violations	"output" dependences
"wread" violations	"anti-" and "flow" dependences
"near-violations" (not emphasized)	"non-loop-carried" dependences (full)
"alpha" references	"source" references
"beta" references	"sink" references

Table 7.1 TRAINING terminology differences

pares how the terminology used in training differed for each group. In the document, they were advised that they were all working different problems using different procedures and should not be concerned if others seemed to take different amounts of time to complete the experiment. The 24 regular problems representing unique combinations of CARRIED (2 levels), DEPTYPE (3 levels), ANNOTATION (2 levels), and REVPAR (2 levels) were organized into a set of twelve programs each containing two problem loops. A given problem was initially presented in the context of both the original and transformed programs. Fig. 7.3 is an example of an **Xbro***wser* screen for this "Comparison" stage of a problem taken from a problem set generated for but not used in the experiment. Subjects used either the "Slicer" button for Manual problems or the "Annotater" button for Annotated problem as in the figure to continue to the displays for a 16-item checklist designed to isolate array elements ultimately affected by a dependence, if any. Checklist displays showed a minimal version of the program similar to a slice as defined by Weiser [Wei82, WL86] containing only statements relevant to the current problem's loop in the upper left with optional annotations as in this example, an automatically updating set of instructions in the upper right for filling in the checklist, a work sheet of editors for the checklist in the lower left, and a set of yes/no questions in the lower right; the questions were concealed until the work sheet for the checklist was completed. Fig. 7.4 shows the last step of the checklist for the same example problem.

After completing the regular problems, subjects answered an extensive on-line questionnaire including standard Likert scales covering many topics possibly relevant to performance on sequential programming tasks. [Par50, Ham84] For example, there were questions covering high school and college mathematics and computer science

Figure 7.3 Comparison Displays for first problem of problem set 5.3

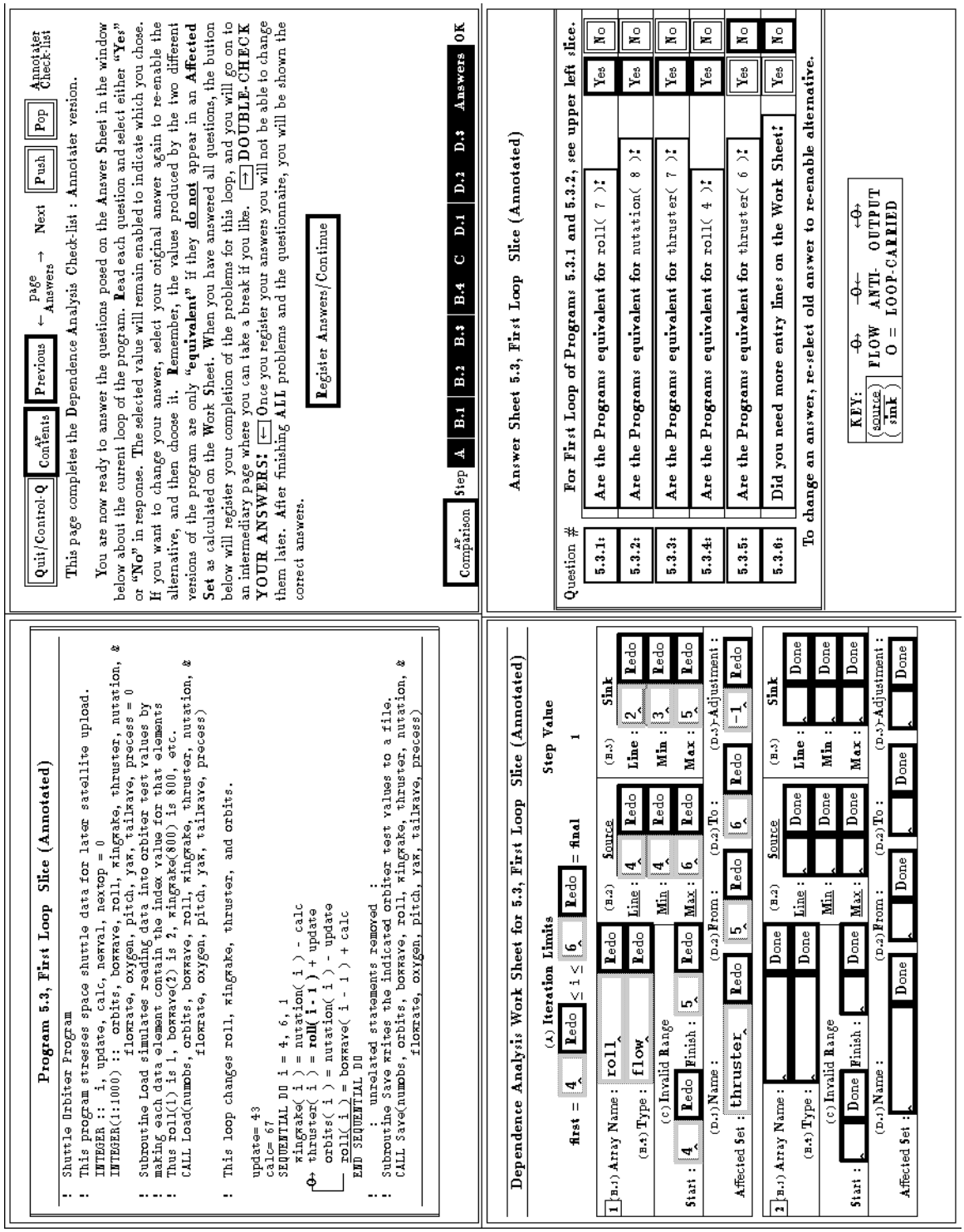


Figure 7.4 Problem Displays for Step ANS
of first problem of problem set 5.3

performance and background [GS89, KLS85, WHN93], programming experience, SAT and GRE scores [GS89], and measures of confidence in programming ability and enjoyment of working problems [WHN93]. In all, eighteen question areas provided more than seventy measures for consideration. The complete text of the questionnaire may be found in Appendix B. Besides acting as a check on the assumption of the comparability of the TRAINING groups, the questionnaire data were used for *post hoc* correlations with subject performance averages described presently.

Subjects took between 2.7 and 5.6 hours to complete the entire experiment, involving almost 800 displays and filling in 300 editors, and including an online debriefing that explained the difference between the TRAINING groups and offered an opportunity to learn the alternative method. The average time was 3.5 hours, and the median time was 3.2 hours.

7.5 Results

For both error severity and time to complete the problem checklist, the main technique employed was analysis of variance (ANOVA), with TRAINING (2 levels) treated as a between-subjects variable while CARRIED (2 levels), DEPTYPE (3 levels), ANNOTATION (2 levels), and REVPAR (2 levels) are treated within-subjects to control for individual variance, using appropriate denominator terms of classical ANOVA as suggested by McCall [McC86] and Winer [Win71].

7.5.1 Error Severity Analyses

The 26 subjects – 16 in the Algebraic Formulation and 10 in the Dependence Analysis TRAINING group – worked 24 problems covering all possible combinations of the within-subjects experimental variables CARRIED (2 levels), DEPTYPE (3 levels), ANNOTATION (2 levels), and REVPAR (2 levels). For each problem, mistakes made were scored as "ERROR", based on a scale from 0 to 8, with 0 being error-free performance, 1 being inconsequential procedural or typographical error *etc.* up to 7 when affected arrays were misidentified, and 8 when the CARRIED nature of a dependence was misdiagnosed. Fig. 7.5 compares the relative frequencies of each class of ERROR score for each TRAINING group for CARRIED. Notice that Non-loop-carried problems only produced ERROR scores of 0, 1, and 8 since successful completion only required entering two values and attesting to the lack of a Loop-

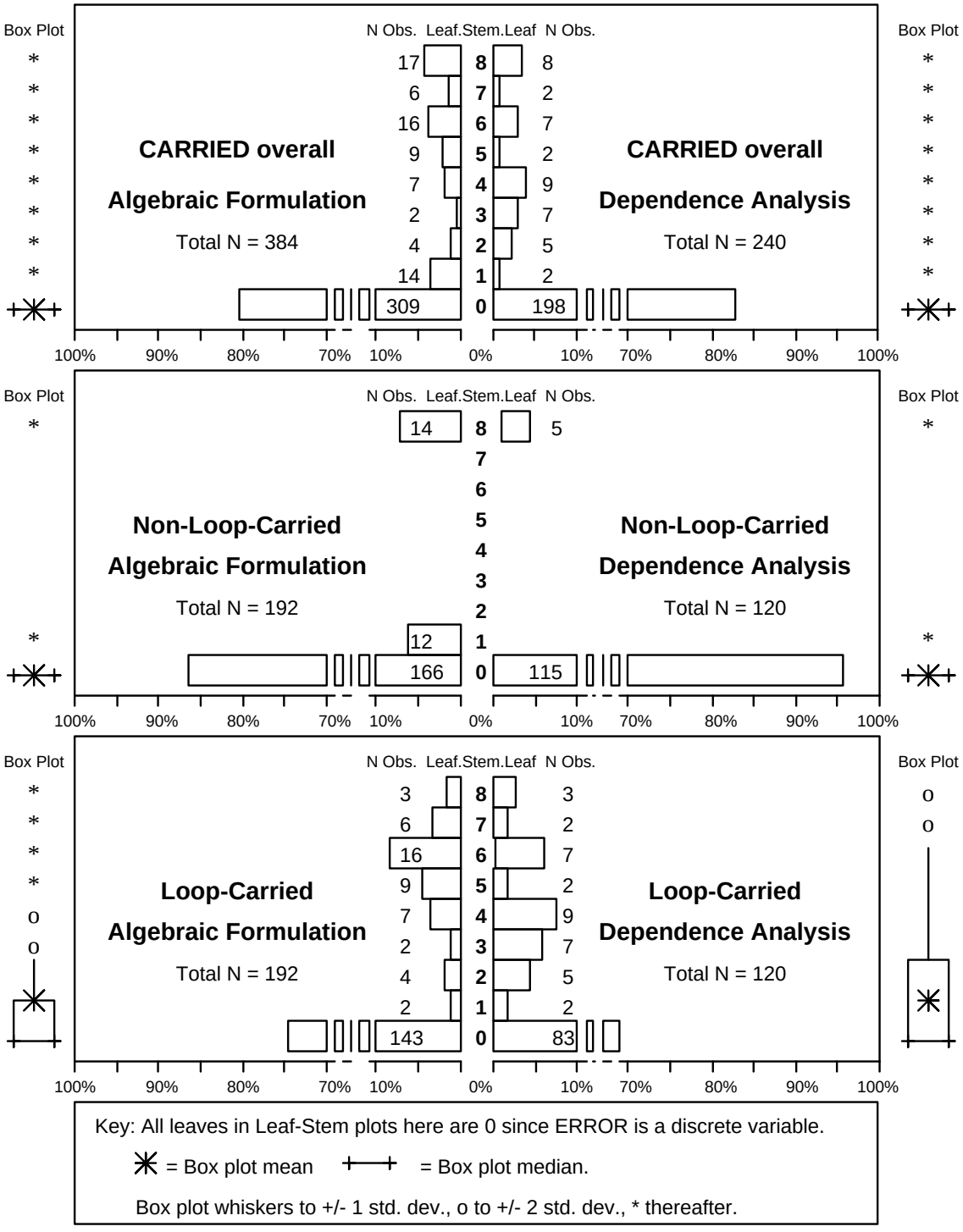


Figure 7.5 ERROR severity Stem-Leaf and Box Plots by TRAINING for CARRIED and its levels

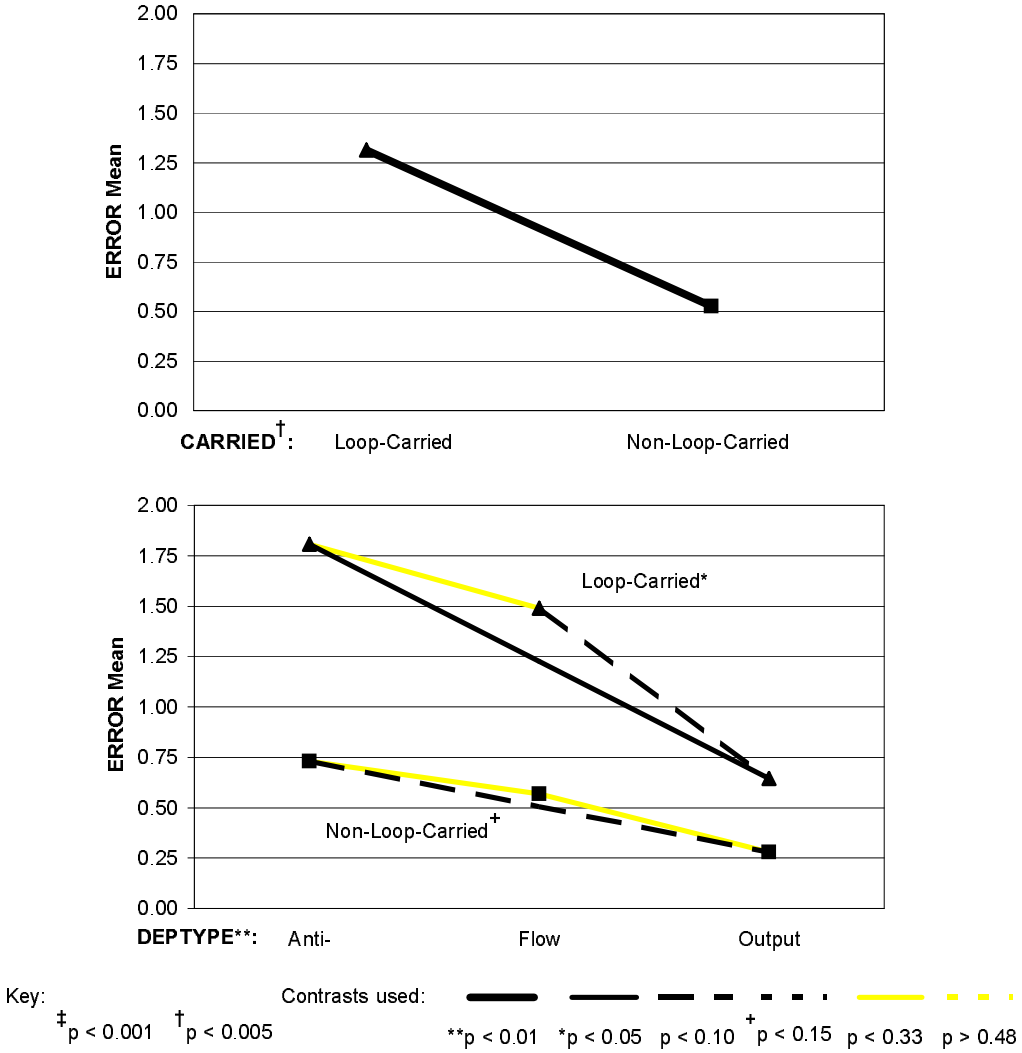
Carried dependence. Thus ANOVA was performed both on the combined error data and separately for each level of CARRIED.

Results for ERROR analysis are summarized in the graphs of Fig. 7.6. In the CARRIED-included ERROR analysis greater opportunities for error in Loop-Carried problems resulted in more severe errors on average than Non-Loop-Carried problems. Additionally, differences were found between Anti-, Flow, and Output dependence problems with Anti- problems weakly worse than Flow problems and very strongly worse than Output problems, with Flow problems somewhat worse than Output problems.

For the CARRIED-separate ERROR analyses the restricted range of error opportunities for Non-Loop-Carried problems seemed to limit variance as well: while the order of ERROR means for the levels of DEPTYPE is the same as for CARRIED-included and Loop-Carried analyses (Anti- > Flow > Output) they are more compacted. Even Anti- and Output dependence problems are only somewhat distinguishable, while the other relationships are poorly discriminable at best. In contrast the Loop-Carried analysis for the combined analysis finds Anti- dependence problems only poorly discriminable from Flow problems, but Flow problems somewhat worse than Output problems while Anti- problems are strongly worse than Output problems.

ANOVA results in the CARRIED-included ERROR analysis for the CARRIED term have $F^1_{24} = 8.7$, with $p < 0.01$. For the DEPTYPE term in that analysis $F^2_{48} = 5.4$, with $p < 0.01$. No other main or interaction term of the CARRIED-included ANOVA has a significance level less than 0.05. A complete set of F-statistics may be found in Appendix B. Contrasts between each pair of levels of DEPTYPE shown in the figure are from ANOVA without the uninvolved dependence type: Anti- *vs.* Flow overall with $F^1_{24} = 2.5$ and $p = 0.13$, Flow *vs.* Output overall with $F^1_{24} = 3.5$ and $p = 0.07$, and Anti *vs.* Output overall with $F^1_{24} = 8.4$ and $p < 0.01$.

Of the other experimental factors, ANNOTATION has $F^1_{24} = 2.4$ and $p = 0.13$, REVPAR has $F^1_{24} = 1.07$ and $p = 0.31$, and TRAINING has $F^1_{24} = 0.09$ and $p = 0.77$; the result for ANNOTATION suggests a possible weak benefit for Annotated problems indicating further empirical work or another form of annotation is needed, but the other two are convincingly negative as far as ERROR is concerned. [Coh88] The weakness of the ANNOTATION result is probably due to the low overall error rate compounded by the fact that for Non-Loop-Carried Annotated problems, the intentional absence of a graphical dependence annotation was overlooked in six instances, resulting in severe error ratings that might have been avoided if another form of an-



Note marks denote significance level for the associated term: CARRIED overall in the upper graph, DEPTYPE overall, Loop-Carried DEPTYPE, or Non-Loop-Carried DEPTYPE in the lower graph. Graph contrasts denote significance level for the various contrasts, concentrating on Anti- *vs.* Flow, Flow *vs.* Output, and Anti- *vs.* Output in the lower graphs with Loop-Carried above and Non-Loop-Carried below.

Figure 7.6 ERROR severity results for CARRIED and DEPTYPE levels

notation had been used for Non-Loop-Carried dependences, as is optionally available in the ParaScope parallel programming environment.

Results of the ANOVA for the *a priori* interaction term hypotheses give for CARRIED-included CARRIED×DEPTYPE the ratio $F^2_{48} = 1.15$ and p= 0.33, for TRAINING×DEPTYPE $F^2_{48} = 0.13$ and p= 0.88, and for TRAINING×CARRIED $F^1_{24} = 0.35$ and p= 0.56; these results all fail to support the hypothesized interactions.

For the separate Non-Loop-Carried error analysis, no result has a significance level less than 0.05: DEPTYPE is only reported here in contrast to the results for Loop-Carried analysis, with an overall $F^2_{48} = 2.2$ and p= 0.12. Most of this is from the Anti- vs. Output contrast, which for Non-Loop-Carried problems has $F^1_{24} = 3.3$ and p= 0.08, while Flow *vs.* Output has only $F^1_{24} = 1.5$ and p= 0.23, and Anti- *vs.* Flow only $F^1_{24} = 1.1$ and p= 0.30.

For the separate Loop-Carried error analysis, the only result with significance level less than 0.05 is for DEPTYPE, with $F^2_{48} = 3.97$, with p= 0.025. Contrasts for DEPTYPE are Anti- *vs.* Flow with only $F^1_{24} = 1.06$ and p= 0.31, Flow *vs.* Output with $F^1_{24} = 3.4$ and p= 0.08, but Anti- *vs.* Output with $F^1_{24} = 6.0$ and p= 0.02.

7.5.2 Time Analyses

The most severe ERROR scores affected the time subjects took to complete the checklist task (TASKTIME), as when non-loop-carried and loop-carried dependences were mutually confused. For that reason, only subjects whose most severe ERROR score was 6 or less were included in the time ANOVA, with 12 subjects in the Algebraic Formulation and 6 subjects in the Dependence Analysis TRAINING groups. Because of vast differences in time required to correctly complete Non-Loop-Carried problems (43 seconds on average) and Loop-Carried problems (176 seconds on average), no CARRIED-included ANOVA was performed, only separate analyses on each level of CARRIED; to further improve sampling distribution by reducing skew, ANOVA were performed on inverse TASKTIME values, a technique frequently used when time to complete a task is of interest [AC84]. For ease in understanding means are graphed in seconds as well as $\frac{1}{\text{seconds}}$. Of the experimental variables, TRAINING (2 levels) is treated between-subjects, while DEPTYPE (3 levels), ANNOTATION (2 levels), and REVPAR (2 levels) are treated within-subjects. Figs. 7.7 and 7.8 compare the relative frequencies of $\frac{1}{\text{TASKTIME}}$ values for each TRAINING group used in the Non-Loop-Carried and Loop-Carried analyses, respectively.

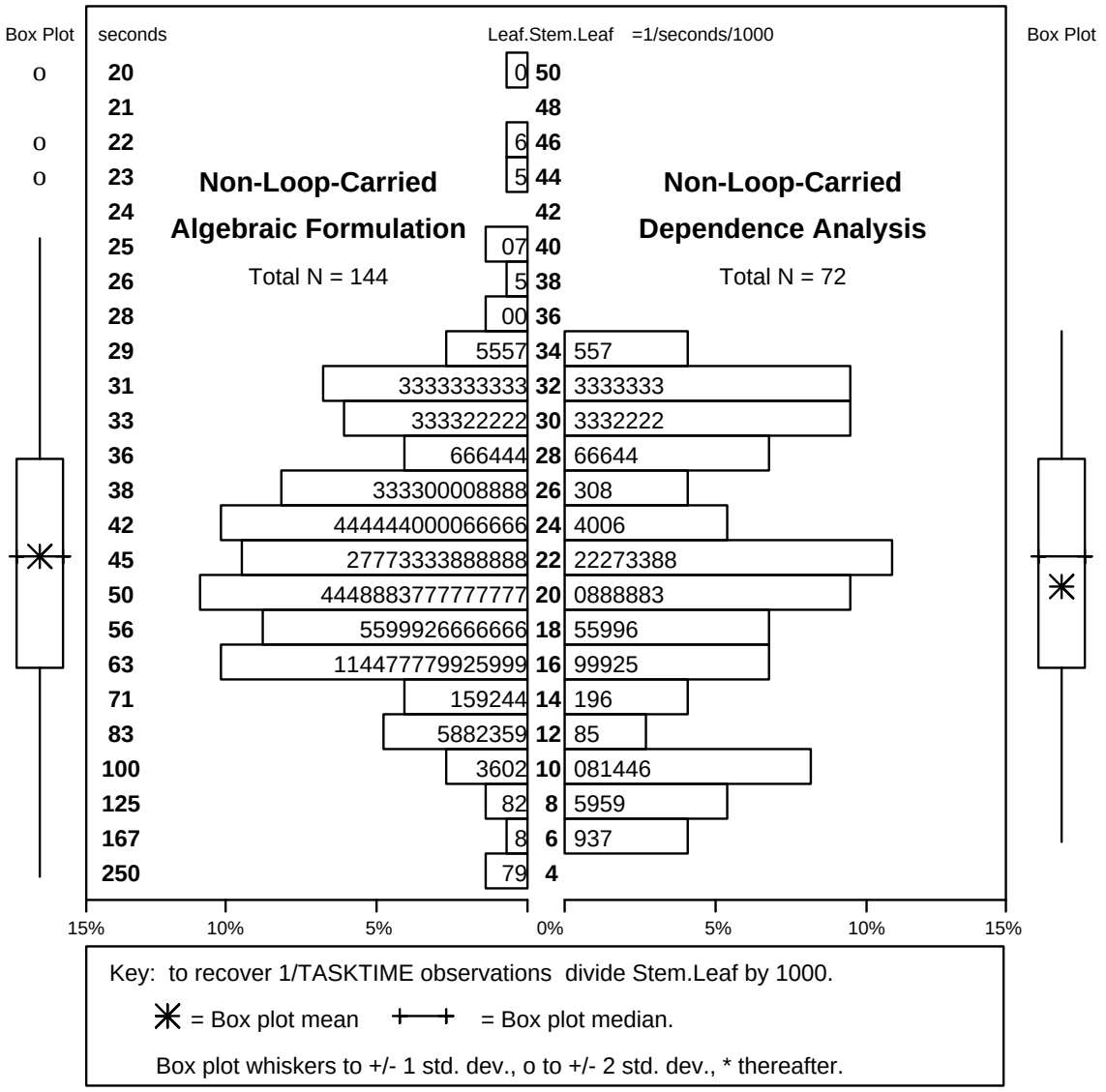


Figure 7.7 $\frac{1}{\text{TASKTIME}}$ Stem-Leaf and Box Plots by TRAINING for Non-Loop-Carried problems

Fig. 7.9 summarizes results for TASKTIME with ANNOTATION and DEPTYPE means and contrasts shown for each level of CARRIED; the indistinct means of Non-Loop-Carried DEPTYPE are included for comparison. For both Non-Loop-Carried and Loop-Carried problems the potential of the Annotated method of ANNOTATION is amply shown, with Annotated problems faster than Manual problems in both cases.

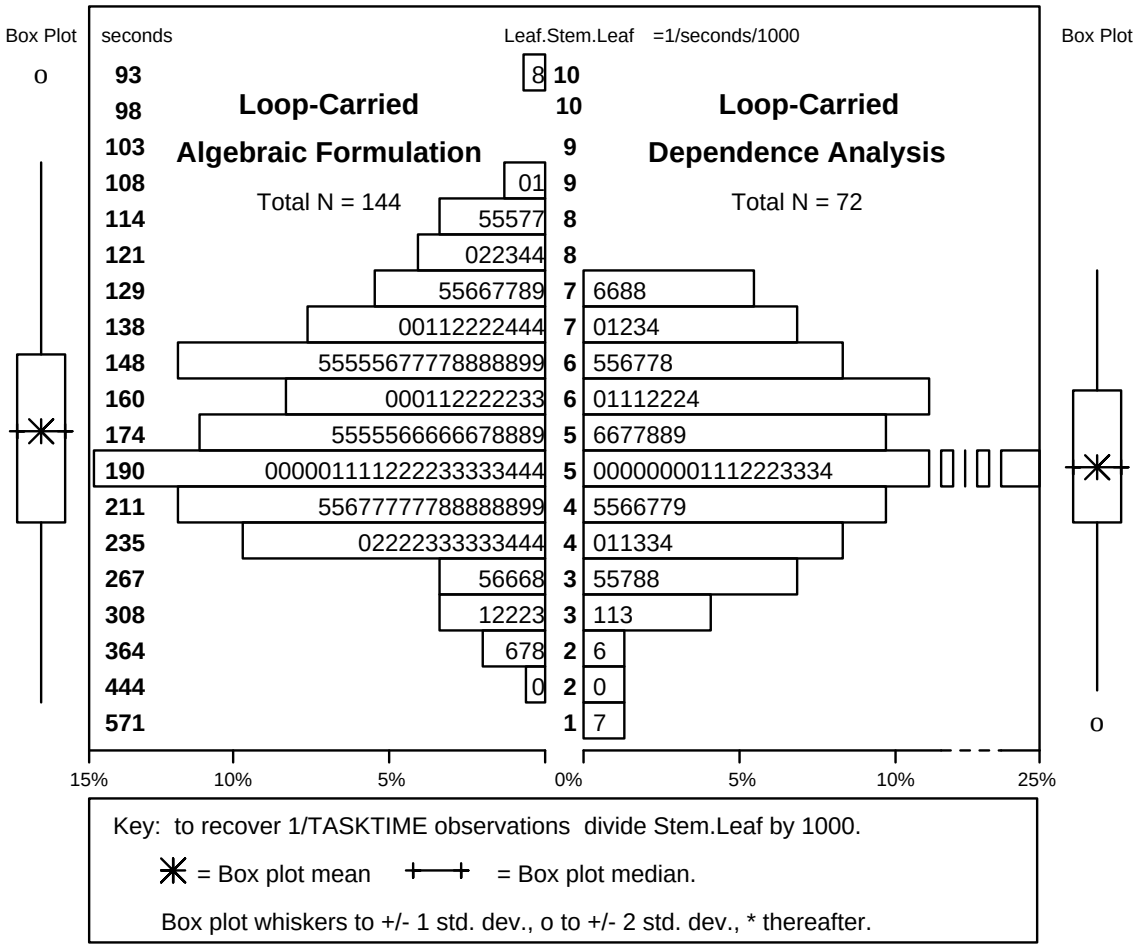
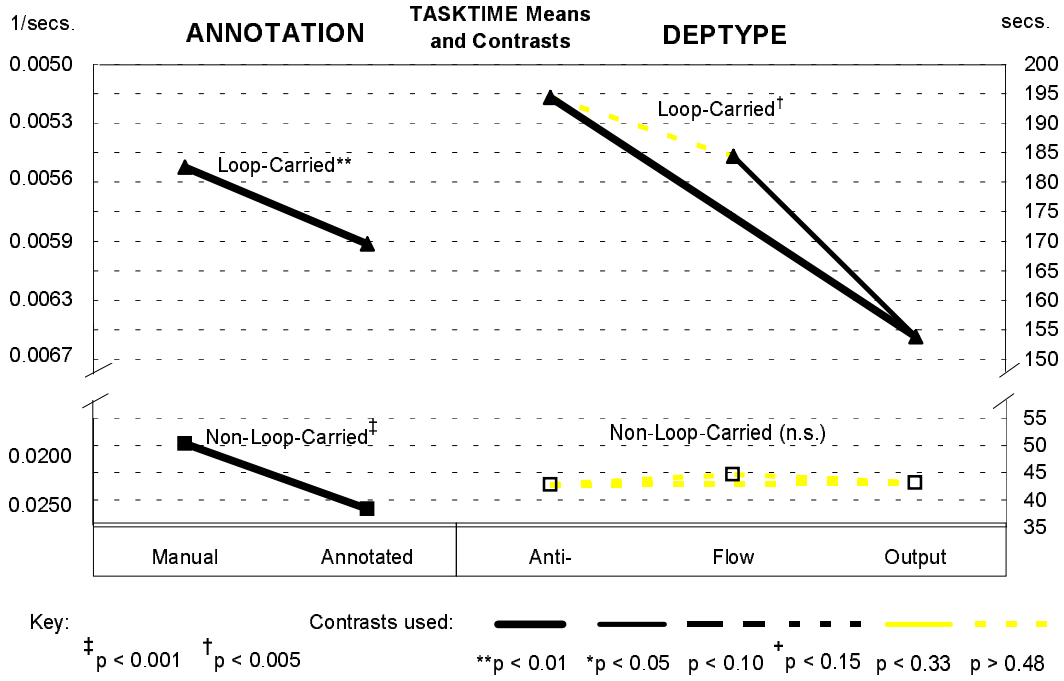


Figure 7.8 $\frac{1}{\text{TASKTIME}}$ Stem-Leaf and Box Plots by TRAINING for Loop-Carried problems

Results for DEPTYPE differ with CARRIED level however: DEPTYPE is very important in Loop-Carried problems, with Anti- dependence problems only a little slower than Flow problems, but Anti- and Flow problems both definitely slower than Output problems. In the Non-Loop-Carried problems however, DEPTYPE does not matter, varying less than two seconds from the fastest to the slowest mean.

For Loop-Carried problems, ANOVA for the ANNOTATION term has $F^1_{16} = 8.7$ and p< 0.01, while for Non-Loop-Carried problems $F^1_{16} = 32.5$ and p< 0.001. For the DEPTYPE term, for Loop-Carried problems ANOVA has $F^2_{32} = 7.6$ and p< 0.01, while for Non-Loop-Carried problems $F^2_{32} = 0.14$ and p$= 0.87$. No other result of ANOVA had a significance level less than 0.05. A complete set of F-statistics may



Note marks denote significance level for the associated term: Loop-Carried ANNOTATION overall in the upper left graph, Non-Loop-Carried ANNOTATION overall in the lower left graph, and Loop-Carried DEPTYPE overall in the upper right graph. Non-Loop-Carried DEPTYPE overall was not significant, with p= 0.87. Graph contrasts denote significance level for various contrasts, concentrating on Anti- *vs.* Flow, Flow *vs.* Output, and Anti- *vs.* Output in the right two graphs with Loop-Carried above and Non-Loop-Carried below.

Figure 7.9 TASKTIME means and contrasts by CARRIED for ANNOTATION and DEPTYPE levels

be found in Appendix B. The contrasts between each pair of levels of DEPTYPE shown in the figure are from ANOVA leaving out the uninvolved dependence type: for Loop-Carried problems, Anti- *vs.* Flow with $F^1_{16} = 0.52$ and p= 0.48, Flow *vs.* Output with $F^1_{16} = 7.3$ and p= 0.02, and Anti *vs.* Output with $F^1_{16} = 11.3$ and p< 0.005, while for Non-Loop-Carried problems Anti- *vs.* Flow with $F^1_{16} = 0.2$ and p= 0.66, Flow *vs.* Output with $F^1_{16} = 0.33$ and p= 0.57, and Anti *vs.* Output with $F^1_{16} = 0.02$ and p= 0.90.

Of the other experimental factors, Loop-Carried problems have for REVPAR $F^1_{16} = 0.03$ and p=0.86, and for TRAINING $F^1_{16} = 1.9$ and p= 0.18, while Non-Loop-Carried problems have for REVPAR $F^1_{16} = 0.49$ and p= 0.49, and for TRAINING $F^1_{16} = 0.82$ and p= 0.38. These results are convincingly negative for REVPAR. For

TRAINING, the possibility of a weak effect in the Loop-Carried case suggests caution. [Coh88]

For the *a priori* interaction term hypotheses, no CARRIED-included ANOVA was done for CARRIED×DEPTYPE and TRAINING×CARRIED, but for the former note that DEPTYPE is significant for Loop-Carried problems ($p < 0.01$) and insignificant for Non-Loop-Carried problems ($p = 0.87$), a strong indication that adverse data-flow and not syntactic appearance matters. For the latter, the result for Non-Loop-Carried TRAINING ($p = 0.38$) suggests that glossing over non-loop-carried dependences has no effect worth further consideration. The last specific interaction term to consider, TRAINING×DEPTYPE, has $F^2_{32} = 0.79$ and $p = 0.46$ for Non-Loop-Carried problems, and $F^2_{32} = 1.36$ and $p = 0.27$ for Loop-Carried problems, suggesting no possible benefit from reducing the distinction between anti- and flow dependences.

7.5.3 Subject Background Information Correlation with Performance

Besides confirming the comparability of the two TRAINING groups, the extensive questionnaire data were used to try to correlate information about the background of subjects to performance on ERROR and TASKTIME measures. These questions were based in part on factors thought to be relevant to performance on sequential programming comprehension as reported by Hammer[Ham84]. Since CARRIED, ANNOTATION, and DEPTYPE all had effects on ERROR and TASKTIME, the questionnaire data were tested against both level of CARRIED separately for all possible combinations of ANNOTATION and DEPTYPE separately and jointly using Pearson product-moment correlation "R" values [SAS90a]. Only correlations that have are significant (with $p < 0.05$), are fairly strong (have $|R| > 0.4$), and fit a larger picture reasonably well (*i.e.*, that make sense in context with related correlations that may not necessarily meet the other two criteria) are reported here, but it is important to remember that the significant error and time effects described previously required 12 averages for both ERROR and TASKTIME be tested for each level of CARRIED for correlation with 38 background factors from the questionnaire, and that these correlations were tested *post hoc*. It is for this reason that the previously mentioned criteria for reporting were selected. A more complete report of the questionnaire correlations may be found in Appendix B.4.

Standardized test scores ...

Question 6.4 in the questionnaire concerned scores on several standardized tests: "Try to recall your scores for any of these standardized tests, if taken: What were your SAT (Scholastic Aptitude Test) scores? Verbal:__ Mathematical:__ What were your GRE (Graduate Record Examination) scores? Verbal:__ Mathematical:__ Analytical:__"

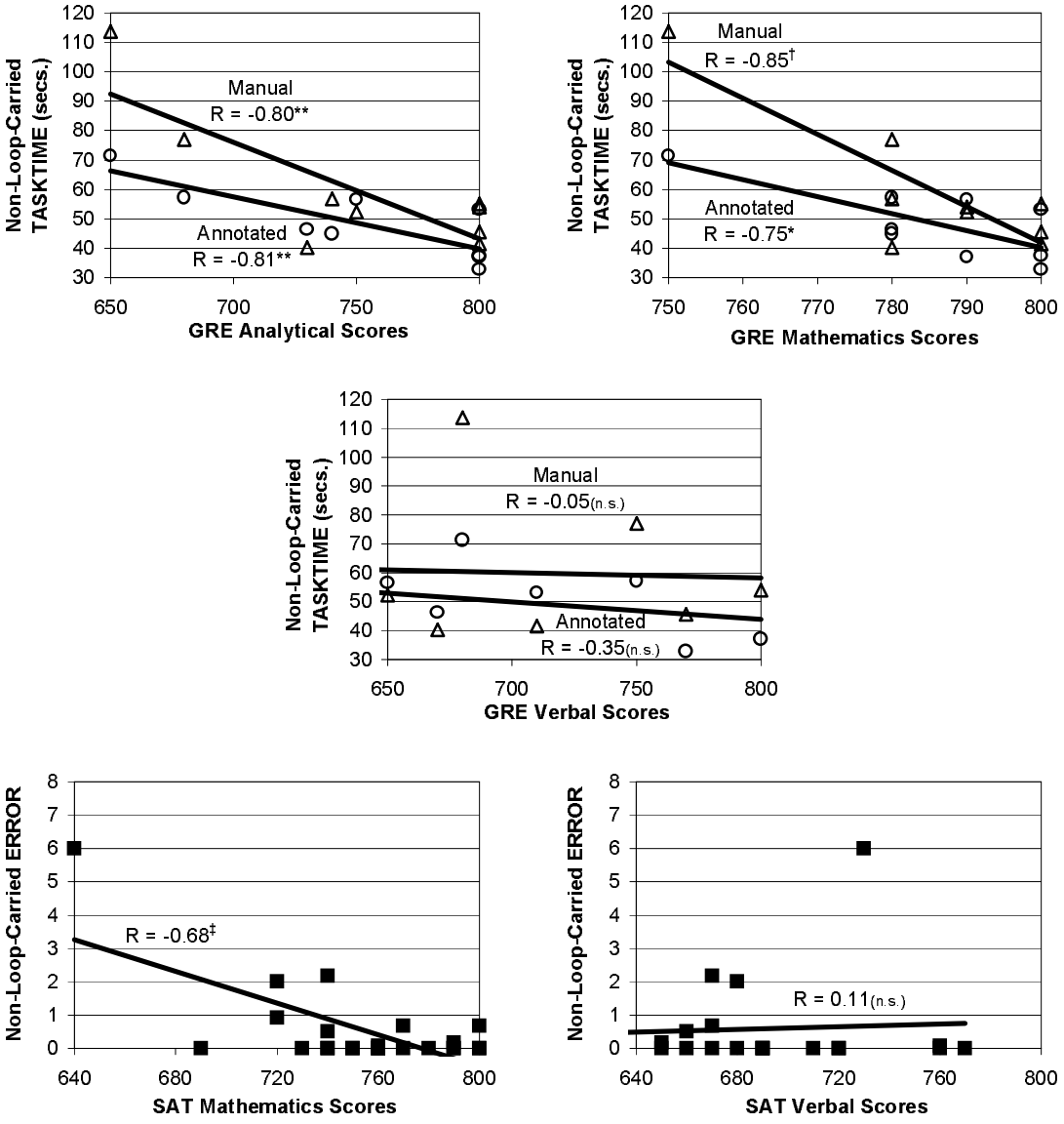
... and ERROR and TASKTIME

Both SAT and GRE mathematical scores were found to have significant correlation with several ERROR and TASKTIME averages, summarized in Fig. 7.10. For Non-Loop-Carried TASKTIME, GRE analytical scores had a strong effect on both Annotated and Manual problems, with $R = -0.81$ and $p < 0.01$, and $R = -0.80$ and $p < 0.01$ respectively. A similar effect held for GRE mathematics scores, with $R = -0.75$ and $p < 0.05$, and $R = -0.85$ and $p < 0.005$ for Non-Loop-Carried Annotated and Manual TASKTIME respectively. Finally, SAT mathematics scores were shown to correlate with improved ERROR performance, with $R = -0.68$ and $p < 0.001$. In contrast, SAT and GRE verbal scores had virtually no influence on these measures: SAT verbal scores correlate to Non-Loop-Carried ERROR with $R = 0.11$ and $p = 0.62$, while GRE verbal scores correlated to Non-Loop-Carried Annotated and Manual TASKTIME values with $R = -0.35$ and $p = .40$, and $R = -0.05$ and $p = 0.69$, respectively.

Mathematics and Computer Science background ...

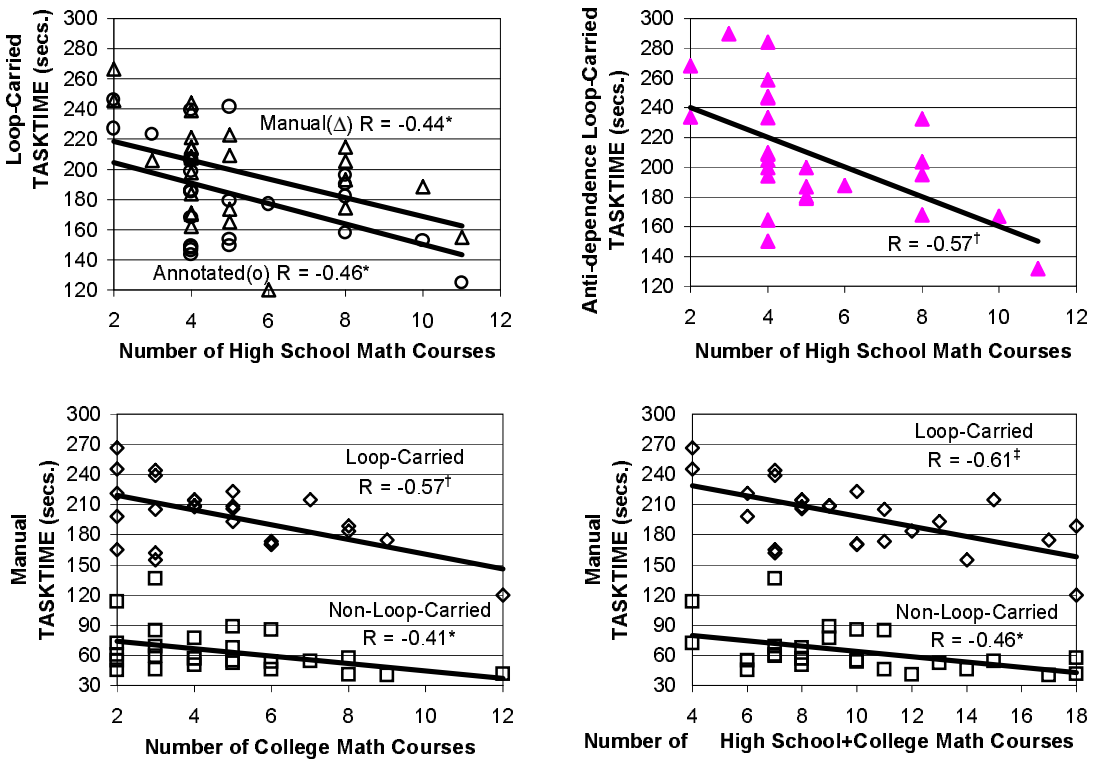
Question 6.5 in the questionnaire was: "How many **computer science** courses have you taken in high school and college? High School:___ College:__ In your estimation, which is closest to your **grade point average** for them: A/B/C/D/F", with grade point averages scored 4.0 to 0.0.

Question 6.6 in the questionnaire was similar: "How many **mathematics** courses have you taken in high school and college? High School:__ College:__ In your estimation, which is closest to your **grade point average** for them: A/B/C/D/F", with grade point average scored 4.0 to 0.0.



Key: ‡p< 0.001 †p< 0.005 ** p< 0.01 * p< 0.05
Note marks in these graphs denote significance level, if any, for the associated correlation.

Figure 7.10 Correlations of TASKTIME with GRE and ERROR with SAT scores



Key: ‡p< 0.001 †p< 0.005 ** p< 0.01 * p< 0.05
Note marks in these graphs denote significance level for the associated correlation.

Figure 7.11 TASKTIME correlations with Numbers of Mathematics Courses

... and TASKTIME

Fig. 7.11 depicts some of the significant correlations between the number of high school and college mathematics courses taken and various TASKTIME measures. The number of high school courses were negatively correlated to both Annotated and Manual Loop-Carried TASKTIME with R= −0.46 and R=−0.44 respectively and p< 0.05 for both. Of the dependence types, the best result was a negative correlation for Anti- dependence Loop-Carried TASKTIME with R= −0.57 and p< 0.005. The number of college mathematics courses was negatively correlated to Non-Loop-Carried Manual TASKTIME with R= −0.41 and p< 0.05 and to Loop-Carried Manual TASKTIME with R= −0.57 and p< 0.005. The total number of high school and college mathematics courses was a similar and even better negative correlate, to

Non-Loop-Carried Manual TASKTIME with R= -0.46 and p< 0.05 and to Loop-Carried Manual TASKTIME with R= -0.61 and p< 0.001.

... and ERROR

Fig. 7.12 shows how estimated grade point averages in high school and college mathematics and computer sciences courses were significantly correlated to various ERROR measures. For Loop-Carried and particularly Anti- dependence Loop-Carried problems, worse mathematics grades were significantly correlated to increased average ERROR with R= -0.43 and p< 0.05, and R= -0.60 and p< 0.001 respectively. For Non-Loop-Carried ERROR, similar significant correlations were found with both mathematics and computer sciences grades with R= -0.64 and p< 0.001, and R= -0.53 and p< 0.01 respectively. The Non-Loop-Carried results are less robust than those for Loop-Carried problems: dropping "C" grade data points affects correlations for Non-Loop-Carried problems more.

Non-programming workload percentage ...

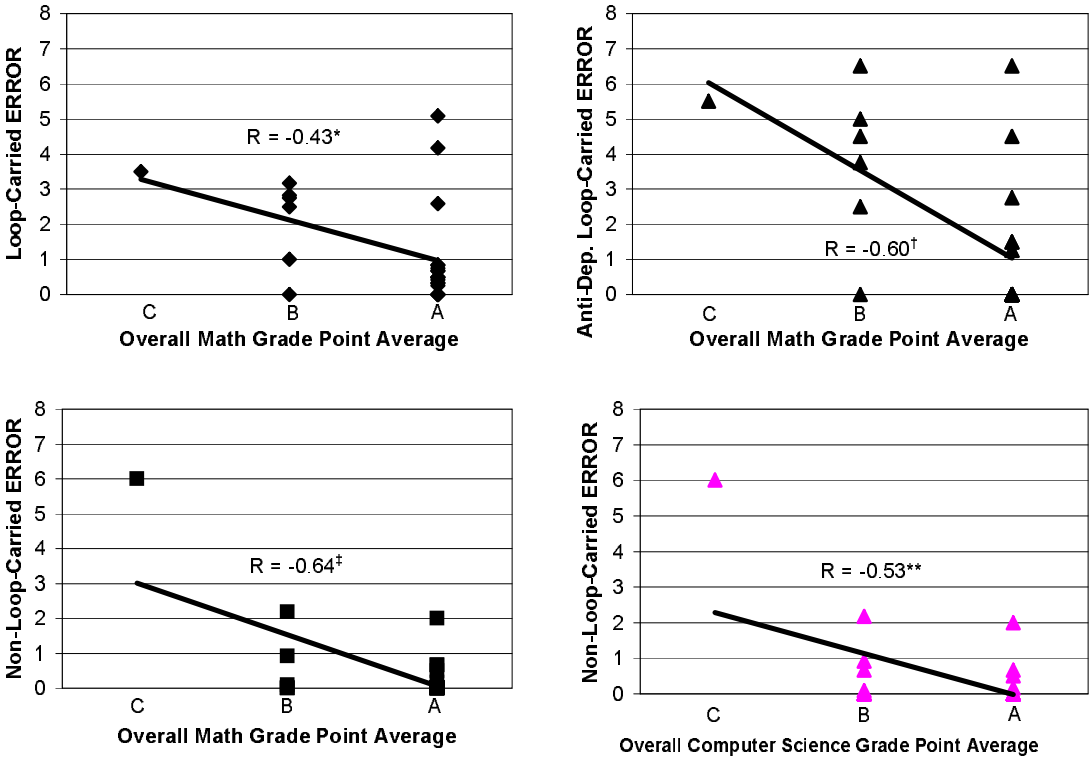
Question 6.13 in the questionnaire was: "What percentage of working time do you spend doing: non-programming-related activities?__ sequential programming?__ parallel programming?__" Responses for parallel programming were negligible, with the remaining two choices summing to 100.

... and ERROR

Non-programming work percentages correlated well to ERROR rates on several measures, with subjects working most commonly at programming doing best. Fig. 7.13 summarizes these results. For Non-Loop-Carried problems, ERROR correlated with non-programming work percentages with R= 0.56 and p< 0.005. For Loop-Carried problems, and particularly for Manual Loop-Carried problems, non-programming work percentages correlated with ERROR with R= 0.48 and p< 0.05, and R= 0.53 and p< 0.01 respectively.

7.6 Discussion

It is not surprising that Loop-Carried problems caused more severe errors and took longer to complete the checklist on average than Non-Loop-Carried problems, given

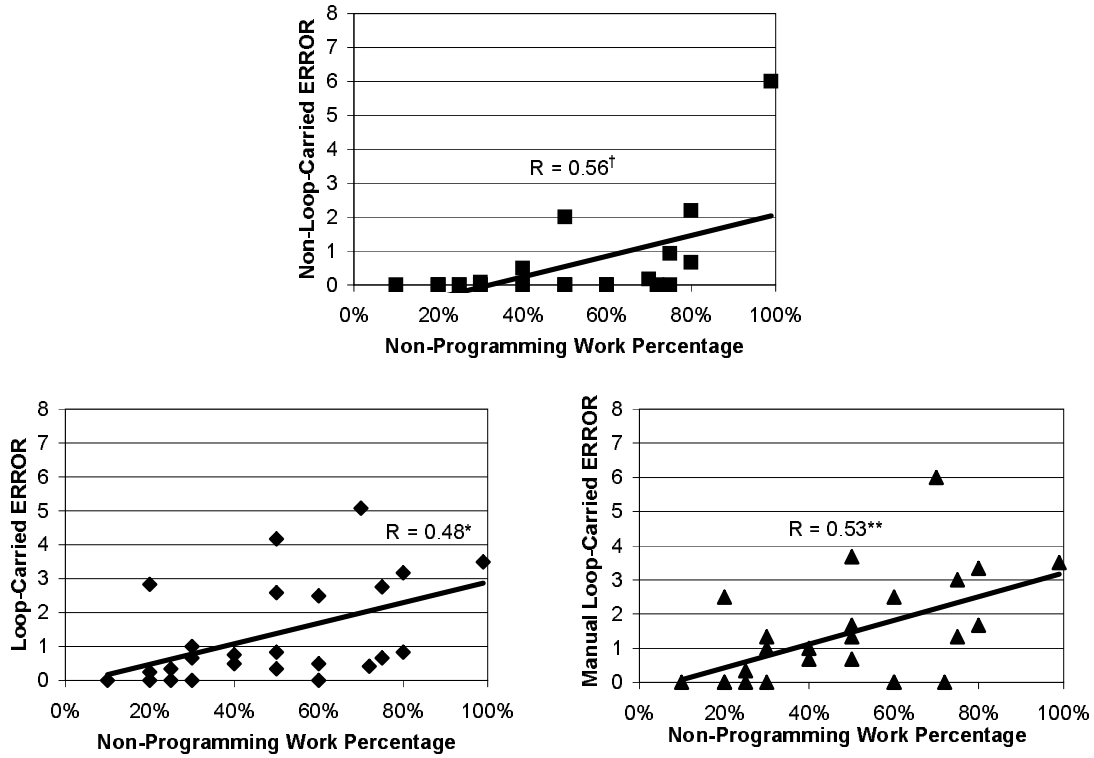


Key: ‡p< 0.001 †p< 0.005 ** p< 0.01 * p< 0.05
Note marks in these graphs denote significance level for the associated correlation.

Figure 7.12 ERROR correlations with Mathematics and Computer Science Grade Point Averages

the experimental design. In the pilot studies, attempts were made to make the checklist more comparable for both levels of CARRIED by requiring subjects to enter null or placeholder values in unused checklist editors, but strong subject resistance prevented use of this design.

The relationships among the levels of DEPTYPE for error severity and for time to complete the checklist for Loop-Carried problems are similar, with Anti- dependence problems worst, then Flow dependence problems, then Output dependence problems. The distinction between Anti- and Flow dependences is much less than between either and Output dependences, suggesting programmers may benefit more from additional assistance with the former two types of dependences, and that more work is needed to determine a relationship between them. The correlations of Anti- dependence



Key: ‡p< 0.001 †p< 0.005 ** p< 0.01 * p< 0.05
Note marks in these graphs denote significance level for the associated correlation.

Figure 7.13 ERROR correlations with Non-Programming Work Percentage scores

Loop-Carried TASKTIME and ERROR with number of high school mathematics courses and mathematics grade point average suggest that a strong background in mathematics may be the best preparation for dealing with the hardest dependences.

The mixed picture for ANNOTATION seems to result from difficulties with one phase of the checklist. While there is no question that Annotated problems were completed more quickly than were Manual problems, the style of annotation used was deficient in two ways. First, the lack of apparent annotations in Annotated Non-Loop-Carried problems was difficult at first for some subjects to recognize: they treated Annotated Non-Loop-Carried problems as Manual Loop-Carried problems, resulting in severe ERROR scores in six cases. Secondly, the checklist required transcription of information about the dependence including type, values of source and

sink expressions at each extreme of the loop, and the array ultimately affected by the dependence. Annotated problems provided only passive assistance with transcription, presenting the information but requiring manual copying of it to the checklist. Examinations of records of the errors seem to show that subjects had more trouble with some parts of transcription than with entering the same information using the Manual method, but that experience with Annotation improved over the course of the experiment for each subject.

Severe errors are clearly not simply the result of haste however, as correlations between TASKTIME and ERROR were positive though weak, with poor time performance generally associated with poor error performance, further validating the conclusions for ANNOTATION and DEPTYPE.

The TRAINING factor had no measurable effect on either error severity or time to complete the problem checklist, showing no immediate benefit from simplification of dependence analysis even in the most basic circumstances such as the minimal problems used here. An experiment using larger programs and loops is suggested to further explore this conclusion.

The lack of significance for REVPAR is even more telling, suggesting that the perception of parallel programming being inherently more difficult than sequential programming is wrong: when the tasks involved are comparable, parallel programming is just another special case of general programming. A simple variation on the current study using the transformed loop in the checklist displays rather than the original loop would be a good way to confirm this further.

Finally, the overall effect of the correlations between subject background information and performance in this study suggest that those programmers with the best backgrounds in mathematics and those working most often at programming itself are most likely to be successful at parallel program comprehension.

Concerning the use of the **Xbro***wser* hypertext system for protocol analysis in this experiment, the ability to successfully process multi-hour-long event-level protocol records and to use dozens of subjects is a persuasive indicator of the value of the system, when common practice in protocol analysis is to use fewer than five subjects or extremely limited tasks [Fis91, CMN83]. **Xbro***wser*'s provision of hypertext and support for computer-assisted instruction coupled with protocol recording proved to be very valuable.

Chapter 8

Summary, Future Research, and Conclusions

In this chapter, I present a summary of the dissertation, the areas of future research that could benefit from the continuation of this work, and some concluding remarks.

8.1 Summary

8.1.1 Introduction

This section summarizes the various chapters of this dissertation to provide context for the sections on future research and concluding remarks. The two major contributions of the research are presented in two parts.

Part I: The Xbro*wser* System

First, I presented two chapters reviewing the several areas related to the design and implementation of **Xbro***wser*, and in a third chapter presented the **Xbro***wser* hypertext system for protocol analysis.

8.1.2 An introduction to protocol analysis

Chapter 1, Protocol Analysis, reviewed some of the history of protocol analysis: much work has been done using both verbal and behavioral protocols. The **Xbro***wser* system is unusual in integrating event-level protocol recording of user interactions with hypertext documents and automated transcription into a general purpose protocol analysis system. No other system has a comparable range of applicability: the WE analysis tools [SSK92] are at present dedicated to study of WE, while SHAPA [SJS89] and PAW [Fis87, Fis91] are essentially limited to verbal protocols and do not support integrated recording and transcription.

8.1.3 Hypertext Issues and Computer-Assisted Learning

In Chapter 2, hypertext issues and computer-assisted learning were discussed. Of the somewhat comparable hypertext systems, **Xbro***wser* is distinguished from **texinfo** and HTML- and SGML-based viewers such as Mosaic and SGML-MUCH [ZR93] by its underlying high-quality pre-formatted text document model, and its support for the needs of computer-assisted instruction, including constrained displays and interactive input coupled with author-controlled session variables. These capabilities allow many models of computer-mediated training to be employed, such as automated mastery learning [Ega88].

8.1.4 Xbro*wser*: Hypertext and Protocol Analysis for User Interfaces

Chapter 4 presented my software system, **Xbro***wser*: a general-purpose hypertext system with support for event-level protocol analysis and computer-assisted instruction. Of the components of **Xbro***wser*, **xbro** is capable of unobtrusively recording user interactions with complex hypertext documents that can provide programmed instruction and simulate many computer situations. The use of high-quality device-independent text documents as an underlying element to hypertext annotations makes **xbro** useful in its own right as a hypertext medium, since text-layout capabilities are limited if not completely lacking in such systems. The protocols collected by **xbro** can be readily translated into forms suitable for summary and statistical analysis using **catevents**. The possible uses of future tools such as **xevents** are also described. The potential value of **Xbro***wser* is demonstrated amply by my experiences with it in the large-scale experiment described there and by the results described in Chapter 7.

The conclusions are that **Xbro***wser* represents a valuable resource in protocol analysis, combining a flexible high-quality hypertext viewer that supports computer-assisted instruction with event-level protocol recording, providing a versatile tool capable of being used in large-scale studies.

Part II: An Empirical Study of Parallel Program Comprehension

The second part covers just such a large-scale study. First, I presented two chapters of work related to the design of the first major empirical study of parallel programming comprehension using **Xbro***wser*, and then the study itself in Chapter 7.

8.1.5 Related work in empirical studies of sequential programming

In Chapter 5, Empirical Studies of Programming, I showed that there is a tremendous body of empirical user-interface research on sequential programming, but virtually none for parallel programming [Wad93]. Studies in the same vein as the current line of research such as Weiser's [Wei82] and Lyle's [Lyl84] experiments on the data-flow technique known as slicing, found evidence for the significant effect of data-flow on sequential program debugging and comprehension. Iselin's study of cognitive effects of various loop and conditional structures [Ise88] on error severity and time to demonstrate comprehension also showed measurable effects on dependent measures analogous to mine. Numerous background factors considered relevant to individual variance in programming ability were described by Hammer in a survey of empirical studies [Ham84]. Several factors used in my background questionnaire were also used by Gowda and Saxton [GS89], while Oman, Curtis, and Nanja [OCN89] studied the effect of programming experience in particular in a study that collected debugging error severity and task completion measures similar to mine. This combined body of work contributed strongly to the design of this research.

8.1.6 Related work in parallel programming

The prospective value of parallel programming was described in Chapter 6, Parallel Programming and Dependence Analysis, together with a major obstacle to parallel programming: data dependences that prevent statements from being successfully executed in parallel. I also introduced an approach to a solution: dependence analysis, capable of detecting such dependences and making them known to programmers who must still bear the responsibility for doing something about them. In particular, the work of Bernstein [Ber66] was particularly useful for the design of the method of Algebraic Formulation, while the design of the method of Dependence Analysis is based in part on Allen and Kennedy [AK87]. The form of graphic annotation used

with the Annotated method of both TRAINING groups is inspired by that used in ParaScope [CCH+88a, CCH+88b].

8.1.7 The empirical study

Chapter 7, An Empirical Evaluation of Dependence Analysis in Parallel Program Comprehension, concluded:

- Annotating program source with graphical dependence arcs describing source, sink, and dependence type as does the ParaScope parallel programming environment improved time required to comprehend loop transformations.
- Loop-carried dependences caused more severe errors on average than non-loop-carried dependences.
- Loop-carried dependences required more time in comprehending loop transformations than did non-loop-carried dependences.
- Anti- dependences caused more severe errors than flow or output dependences, and flow dependences caused more severe errors than output dependences.
- Loop-carried anti- and flow dependences took longer to comprehend than loop-carried output dependences, but dependence type did not seem to matter for non-loop-carried dependences.
- Parallel programming was not shown to be more difficult than an equivalent sequential programming task.
- Simplifying dependence analysis had no significant effect on error severity or time to comprehend loop transformations.

In terms of individual subject background and performance variance controlling for the previous effects:

- Time to comprehend manual and annotated non-loop-carried problems improved with better GRE[1] Analytical and Mathematical scores; no similar improvement was noted for Verbal scores.

[1]Graduate Record Examination

- Error severity on non-loop-carried problems decreased with better SAT[2] Mathematics scores ; no similar improvement was noted for Verbal scores.
- Time to comprehend manual and annotated loop-carried problems improved with the number of high school math courses taken; this was particularly true for anti-dependence problems, while for the number of college math courses and the combined total of high school and college math courses the improvements were even more striking for both loop-carried and non-loop-carried manual problems.
- A better mathematics grade point average correlated to reduced error severity for both loop-carried and non-loop-carried problems and particularly for loop-carried anti-dependence problems; a similar effect was found for computer science grade point average and non-loop-carried error severity.
- Those who indicated spending the smallest percentage of working time on non-programming-related tasks had the lowest error severity rates for non-loop-carried and loop-carried problems; in the latter case, this was particularly true for loop-carried manual problems.

8.2 Future Research

Several directions lie ahead: these may be categorized as improvements to **Xbro***wser*, further investigation of topics begun in the current empirical study, and new research in closely related areas not addressed here.

8.2.1 Improvements to Xbro*wser*

Xbro*wser*, useful as it is in its current form, should be improved further. Several requests have been received for its distribution once it is packaged for general use. Even in its current form **Xbro***wser* provides an admirable tool to tackle these possibilities. As I suggested in Chapter 4, useful additions may also be added to **Xbro***wser*: a more extensive LaTeX interface to the hypertext features of **xbro** coupled with improved display speed and an improved editor user interface standard, an expanded event protocol record for the **catevents** interface, and new tools building on the old such as **xevents**, a protocol event replay and interactive filtering tool. With these, the already dramatic value of **Xbro***wser* can be increased yet more.

[2]Scholastic Aptitude Test

8.2.2 Follow-ups to the current empirical evaluation of dependence analysis in parallel programming

Further experimentation needs to be done to resolve issues relating to the value of annotation in dependence analysis and of alternatives to dependence analysis. These first applications of empirical studies of programming to the field of parallel programming were of necessity highly reductionist, and could not possible begin to cover all possible combinations of factors involved. The significant relationships from the current research that were found between loop-carried and non-loop-carried dependences, among anti-, flow, and output dependences, and for background factors related to mathematics experience and ability and work-related programming habits should be confirmed in larger programs.

Simple variations on the current experimental design could address the failure to find empirical evidence that parallel programming is inherently more difficult than a similar sequential programming task by using the transformed (*i. e.*, the reversed or parallelized) loop rather than the original in the problem displays: if that design affects performance as the current one did not, there would be evidence that the perception that parallel programming is inherently more difficult may be due to performance anxiety or other extraneous influences rather than any intrinsic difficulty in parallel programming. Likewise, the failure to find convincing evidence that graphic annotation of source code with dependence information reduces error severity must be studied further: perhaps another form of annotation or a task less prone to extraneous error could be used.

8.2.3 Other empirical studies related to parallel programming

There are empirical issues beyond those raised by the current study. I mentioned in Chapter 7 that pilot studies suggested that there may be significant problems with extending current programming language constructs for parallelism simply by adding the keyword "parallel", and this should be tested. The empirical approach should also address language constructs other than loops, such as case statements.

8.2.4 Proselytization: Encouraging the use of empirical user interface analysis in parallel programming

Finally, applying empirical user interface research methods to parallel programming should continue until the research community in parallel programming no longer has

to continue repeating the mistakes of past researchers in sequential programming, but can concentrate on the problems uniquely addressed in their own field. Only then will the paradigm of parallel programming be able to assume its position as a major tool for the advancement of computation. With research now beginning in Canada [SWCB93, Wad93] and thanks in part to motivational discussions I made at two of the Empirical Studies of Programmers Workshops, this process has now begun. This dissertation is another major step in that direction.

8.3 Conclusion

In conclusion, the value of the **Xbro***wser* system has been demonstrated by its contributions to proving the relationships that exist between loop-carried and non-loop-carried dependences, between anti-, flow, and output dependences, and between annotated and manual methods of dependence analysis, and to finding correlations between comprehension performance measures and measures of mathematics experience and ability and working habits.

Those relationships have been shown to matter even in the most minimal situations. Now at last, we have a platform on which to build, with a firm foundation.

Appendix A

Experimental Materials Details

A small computer laboratory was used for this study, containing six identical Sun 3/50 workstations configured as X-terminals arranged as shown in Fig. A.1. Workstations included a 19 inch monochrome display of 1024×960 pixels with a 14 inch by 10.75 inch usable screen area, a keyboard, and an optical mouse that could be placed on either side of the keyboard. The workstations were labeled "One" to "Six" as in the diagram and were used in numeric order to avoid overcrowding as much as possible. During the experiment no more than three were ever used at a time, though station Two was unavailable at one point and stations One, Three, and Four were used. On the work-surface by the door were sheets to reserve specific time slots for study participants, and instructions for logging in to **Xbro***wser*. Users were not routinely observed while participating in the study other than through the data collected by

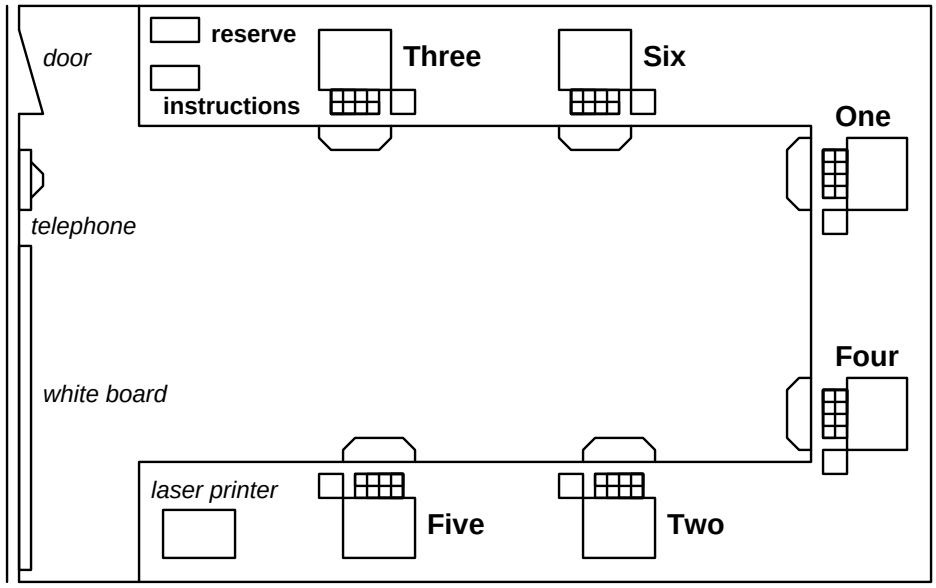


Figure A.1 Diagram of **Xbro***wser* Laboratory

Xbro*wser*, but an experimenter was available for consultation in another office, and could be reached by the telephone provided.

Xbro*wser* was used to present the hypertext programmed-instruction document summarized in Fig. A.2, showing some of the hypertext links used for document navigation. Adjacent boxes were sequentially linked from top to bottom. In general, links branching forward were disabled until a given section was encountered in sequential reading. To simplify the diagram, reference links from the Index and within sections are not shown. Boxes with rounded borders were identical for both TRAINING groups, except those where reference was made to the TRAINING group name, indicated by "*". Dark rectangular boxes highlight material that differed for

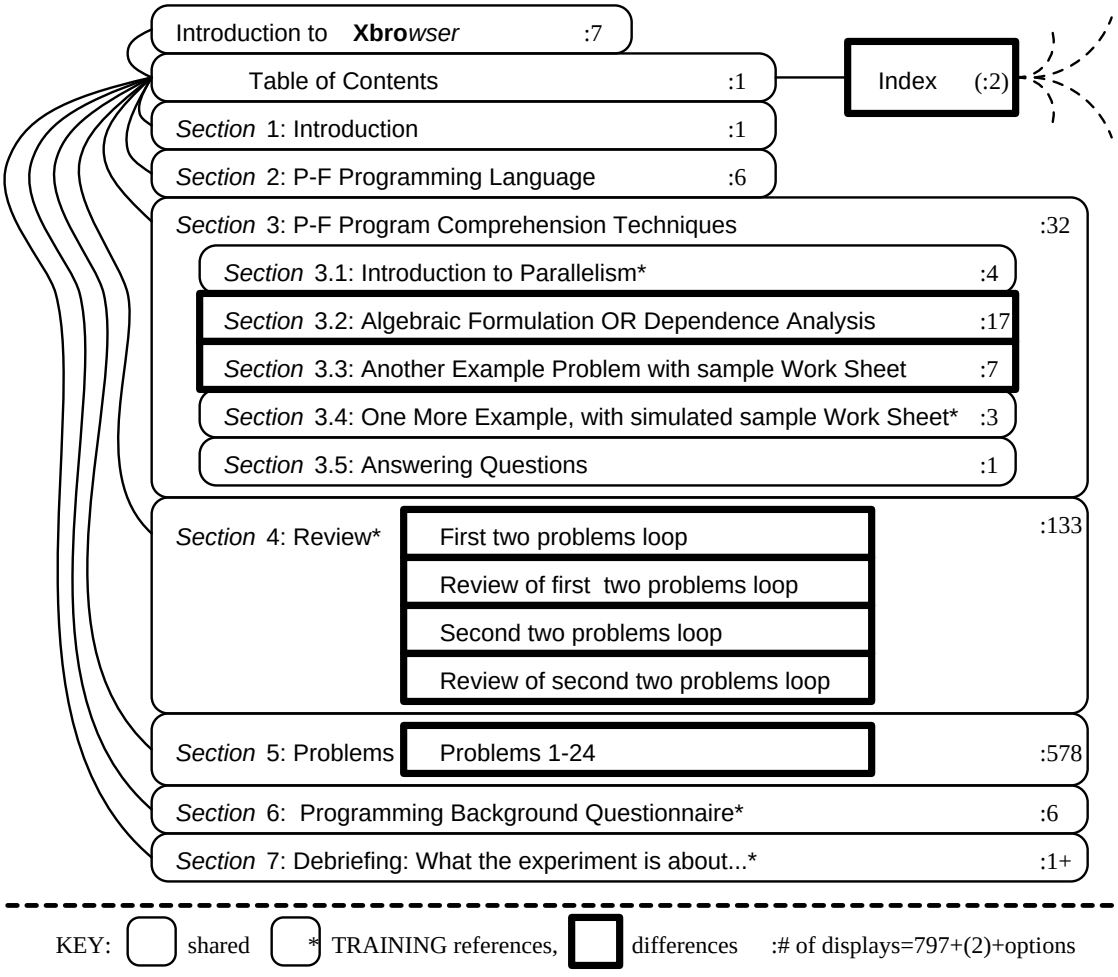


Figure A.2 Overview of the **Xbro***wser* hypertext document used

each group. The number to the right in each box is a summary count of the number of displays used to complete that section of the document: thus the entire process involves 797 displays, not including optional displays such as the two for the Index and post-participation opportunities. Wherever possible, the same text was used for both groups, and when alternative text was necessary for each TRAINING group the number of displays used and the text presented were constrained to be of similar amounts and complexity. The various sections of this document are described in more detail below. To avoid confusion with section references to the current document, section references to the experimental materials are presented in italics.

The review problems used were the same for all subjects except for the differences in TRAINING method. The regular problem sets worked by subjects were generated randomly by choosing problems from a set of templates covering all possible combinations of the experimental factors, randomly choosing a set of template variable names taken from twelve sample programs, and generating random numeric expressions to produce appropriate innocuous or adverse data flow for desired loop behavior. This process produced FORTRAN 77 or C programs with forward and reversed versions of problem loops that automatically generated answer keys for each problem, as well as customized LaTeX code to produce **Xbro***wser* dvi-files for each problem set.

Each subject was assigned randomly to a TRAINING group using a program that also randomly generated twenty-four problems covering all combinations of the other four experimental factors. After viewing the initial scripted demonstration on the use of **Xbro***wser* in which they were advised that records were being kept of their specific interactions with the system, subjects used **Xbro***wser* to participate on their own. **Xbro***wser* maintained records on progress within the experiment and enforced breaks every thirty minutes to reduce fatigue, though breaks could be as short as the subject wished.

The following descriptions present each section of the hypertext document in the order normally encountered, with differences in materials for the two TRAINING groups described in detail. The title of each section is shown with Virtual page numbers used for document navigational reference and window display counts in parentheses if different from page counts.

Introduction to Xbro*wser*: Pages 1-4 (7 Displays)

Subjects learned how to use **Xbro***wser*, including standard "control panels" found on most pages that contain a "Quit" button to exit **Xbro***wser*, a "Contents" button

to go to the Table of Contents, "Previous" and "Next" buttons to move sequentially through the document, a "Push" button to mark the present location in the document, and a "Pop" button to return to a marked location. For navigation, the standard control panel includes the current virtual page number and the name of the current section. Subjects learned to select buttons, use keyboard equivalents, and enter text in editors, as well as the different combinations of on-screen displays used in **Xbro***wser* documents. Before proceeding to the main document, subjects had to enter text correctly in a sample editor.

Table of Contents: Page 5

From the Table of Contents (reachable from any standard control panel throughout the document) subjects could return to the beginning of any section that they had reached sequentially, and could visit the "Index" pages to branch to topics explained throughout the document.

Index: Pages INDEX.1-INDEX.2

The Index contained two pages of terms defined throughout the document, with hypertext buttons to visit the appropriate pages that were only enabled after the pages had been visited sequentially. Certain terms were different for Algebraic Formulation and Dependence Analysis, as discussed below.

Section **1: Introduction:** Page 6

This page introduced and outlined the procedures used in the study. Subjects were advised that they were working on different problems using different procedures and not to be concerned if others seemed to finish at different speeds.

Section **2: The P-F Programming Language:** Pages 7-12

This defined the syntax and semantics of the **P-F** programming language designed explicitly for the problem sets used in this study. P-F is a subset of FORTRAN 90 [BGA90] extended for parallelism by the addition of two keywords used with DO loops: "PARALLEL" for parallel loops, and "SEQUENTIAL" for ordinary loops; the latter was added to even the cognitive load of reading sequential and parallel programs, and to lessen potential problems with similar surface syntax noticed in pilot studies. The six types of statements used in P-F included comments, variable and array declarations with initialization, assignment statements, subroutine CALL statements used to simulate reading and writing data, SEQUENTIAL DO loops, and PARALLEL DO loops. The use of "&" to continue long lines was also supported.

The section on SEQUENTIAL DO loops mentioned that reversing a loop does not always produce the same results, while the section on PARALLEL DO loops referred to the equivalent problem for parallelizing loops.

Section **3: P-F Program Comprehension Techniques (5 subsections):**

Section **3.1: Introduction to Parallelism:** Pages 13-16

This began with a general discussion of the difficulties of correct parallel programming, including Bernstein's conditions, similar to parts of Chapter 6 of this dissertation, and then discussed parallelism in P-F and the relevance of reversing loops to parallelizing loops.

Section **3.2:** ⟨ **Algebraic Formulation** **OR** **Dependence Analysis** ⟩ : Pages 17-31 (17 Displays)

This (including the title) introduced specific terms and procedures associated with the TRAINING method, either Algebraic Formulation or Dependence Analysis. For the former, the concepts of Bernstein's conditions were reinforced and formalized, with terminology designed to balance that of Dependence Analysis as follows:

Algebraic Formulation	**Dependence Analysis**
"violations"	"loop-carried dependences"
"wwrite" violations	"output" dependences
"wread" violations	"anti-" and "flow" dependences
"near-violations" (not emphasized)	"non-loop-carried" dependences (full)
"alpha" references	"source" references
"beta" references	"sink" references

Beside the more comparable alternative terms, the primary differences of Algebraic Formulation materials from those of Dependence Analysis were the simpler treatment of anti- and flow dependences as special cases of a general "read-write" violation of Bernstein's Conditions, and the lack of emphasis on the equivalent of non-loop-carried dependences, referred to as "near-violations" in passing: otherwise the versions were similar for both TRAINING groups. Subjects were also told that they would be working problems using either of two experimental tools under development: a loop "**Slicer**" that lets the user view only the statements from a program relevant to the variables set in a given DO loop in a P-F program, or a loop "**Annotater**" that augmented the Slicer by annotating the loop body with graphical arcs providing in-

formation about any Loop-Carried dependences that were present. Two alternative checklists – "Manual" for the Slicer, or "Annotated" for the Annotater – were provided, giving a step-by-step reference for methods to fill in a "Work Sheet" of editors that determined the precise set of array elements ultimately affected by dependences (if any) in a given problem. For review problems, all thirty editors on a Work Sheet could be used; for regular problems, no more than sixteen were used.

Fig. A.3 outlines the steps used in the checklist, including the Work Sheet editors set and used by each step. Where steps differ owing to differences in TRAINING groups, the word "TRAINING" appears at the left of the step's box. If a step makes use of the violation/dependence arc displayed for Annotated problems, "*" appears in the EDITORS USED column to indicate that some editors listed were replaced or assisted by that use. Fig. A.4 – greatly reduced here to fit – compares part of the displays for the Annotated loops as shown to each TRAINING group. Note the variation in terminology and appearance resulting from the differing treatment of flow, anti-, and non-loop-carried dependences. Here, the example loop contained a loop-carried output dependence and a loop-carried flow dependence.

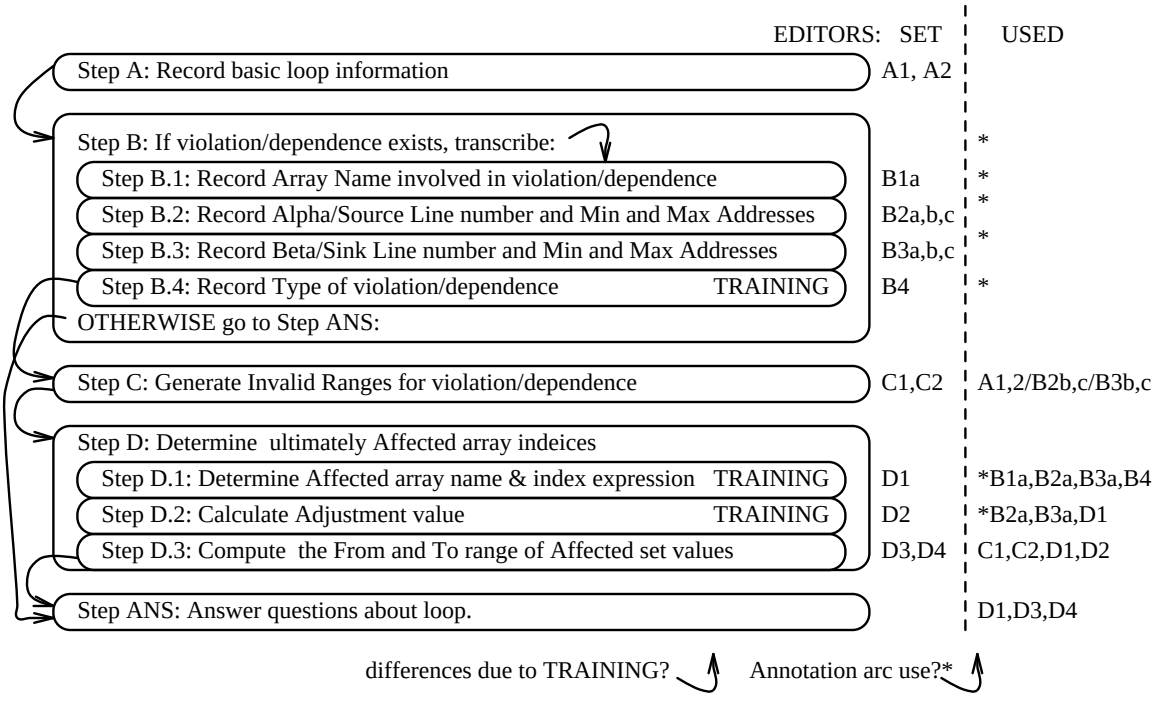


Figure A.3 Outline of Annotated and Manual Checklists

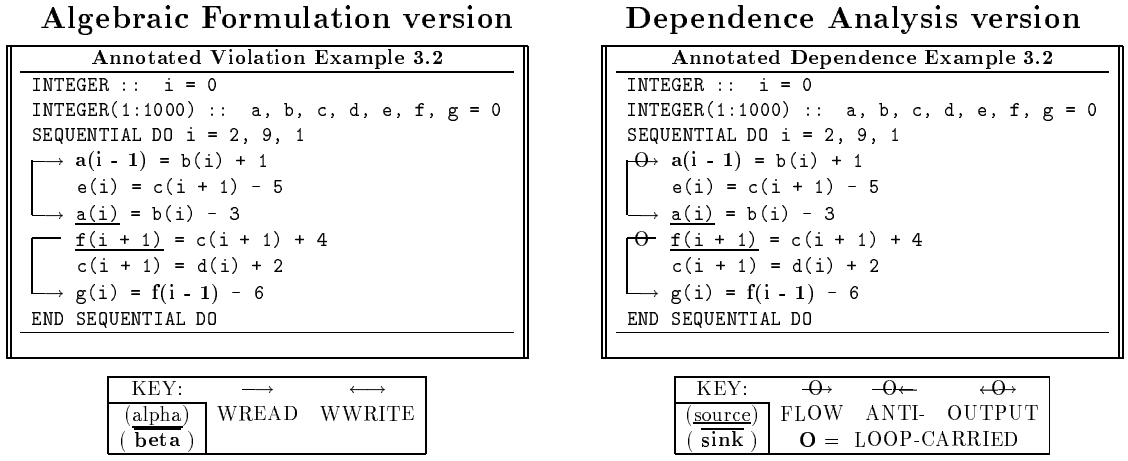


Figure A.4 Annotated checklist-specific TRAINING example

Section **3.3: Another Example with sample Work Sheet:** Pages 32-36 (7 displays)

Subjects were first taught the Annotated method, but the Manual steps required for the same dependences were also shown. Subjects had to enter the correct information – provided on-screen with the appropriate reasons for each step – in a sample Work Sheet for each step of the checklist before being allowed to proceed to the instructions for the next step. Fig. A.5 continued the programmed instruction with a sample Manual problem 3.3. In this example there were three dependences in the loop: non-loop-carried output and flow dependences that required no checklist data entries, and

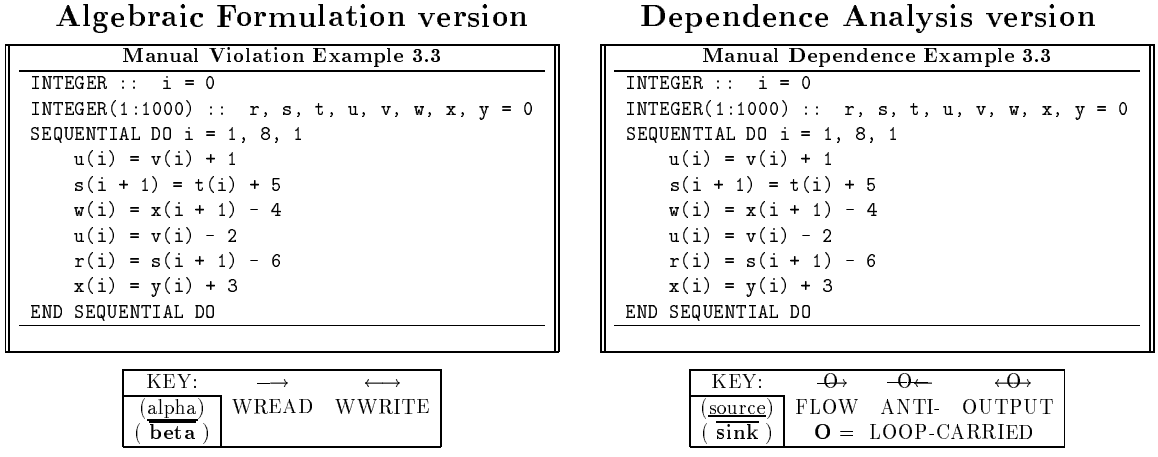


Figure A.5 Manual checklist-specific TRAINING example

a loop-carried anti-dependence. For this second example, subjects were able to enter values before being told the correct answers. If correct, the screen indicated success; if incorrect, the next page contained correct answers and required subjects to correct mistakes before proceeding.

Section **3.4:** **One More Example with simulated sample Work Sheet** : Page 37 (3 displays)

The final example reinforced the importance of watching the title display for program loops: if the word "Annotated" appeared and there were no dependence/violation arcs displayed, no loop-carried dependences/violations existed and the subject could skip the remainder of the checklist. In one pilot version of the study, subjects were required to page through the unused portions of the checklist rather than proceeding directly to answer the questions; while this made the time values for Loop-Carried and Non-Loop-Carried problems more comparable, every pilot subject expressed dislike of the wasted time and so the shortcut was introduced in the last pilot study and was retained in the final version. This shortcut was thus used in Example 3.4, containing no dependences of any form, as seen in Fig. A.6. The corresponding Work Sheet for this example would look something like the simulated sample shown in Fig. A.7 taken from the instructional material and reduced here to fit. The Algebraic Formulation version was identical except for the TRAINING method name in the title area above the work sheet. Note that space is provided for up to two loop-carried dependences to be recorded: the Review problems discussed below had zero, one, or two loop-carried

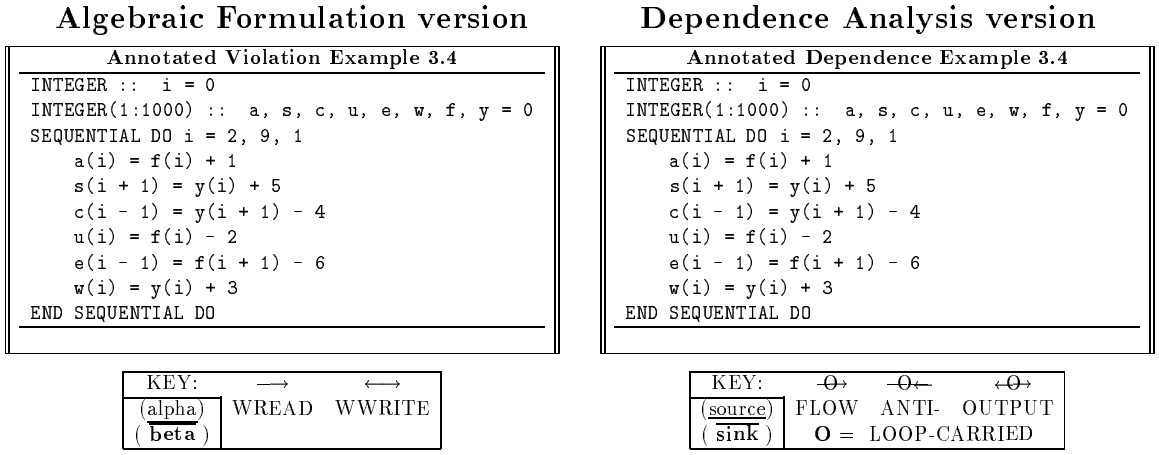


Figure A.6 Annotated example with *no* dependences

dependences per loop, while each of the twenty-four regular problems covering all combinations of the within-subjects experimental factors contained at most one – to take advantage of "over-training" effects.

Section **3.5: Answering Questions:** Page 37

This explained how to answer questions about the transformed program loops used in problem sets, and reassured subjects that the examples used were carefully chosen so that reversing or parallelizing loops would fail in exactly those instances detected by using the checklist for the TRAINING method that they had learned.

Section **4: Review with Sample Problems:** Pages 38-42 (133 displays)

This *section* contained two sample problems sets identical to the type that subjects

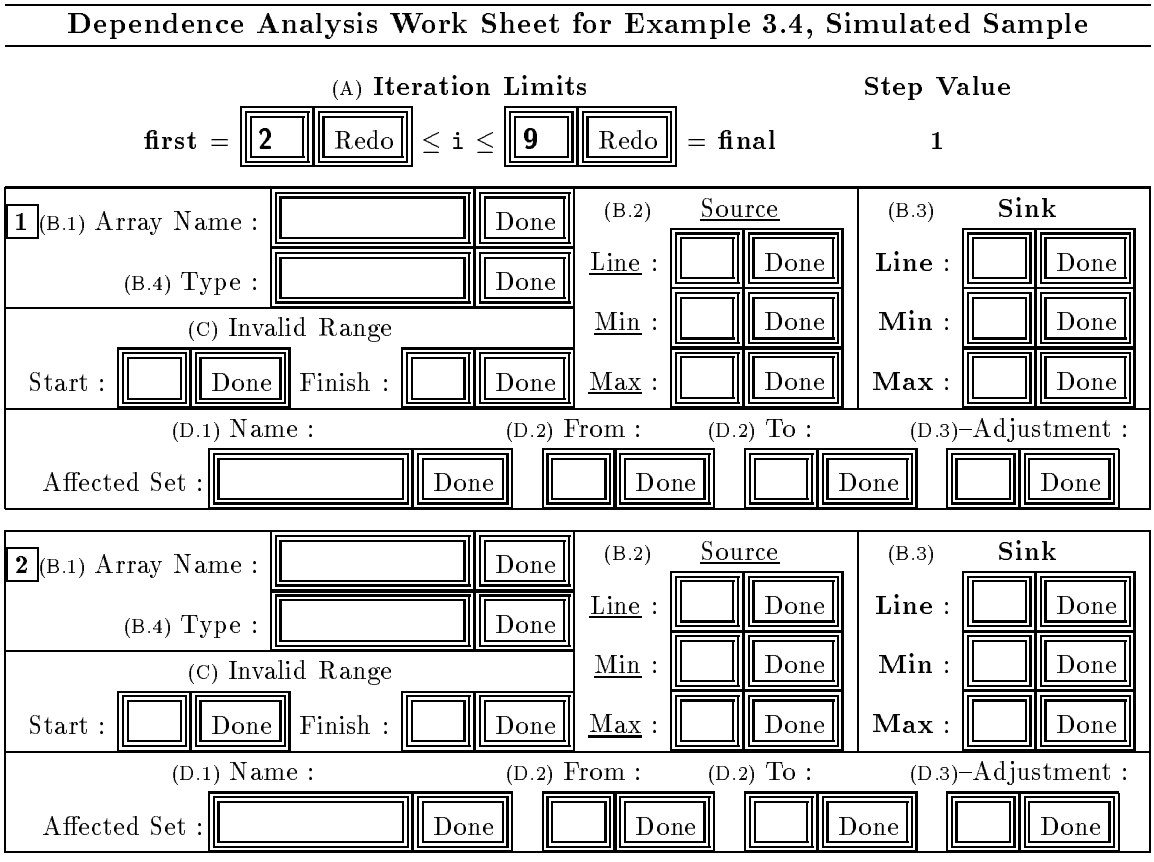


Figure A.7 Simulated example Work Sheet for problem with *no* dependences

would be using later in the study. Each problem set consisted of a program containing two problem loops that were processed individually using either the Slicer or the Annotater. A given problem began by displaying the entire program in two versions: the first used sequential forward DO loops, while the second contained version of the loops that had been either reversed or parallelized. Fig. A.8 is an example of the arrangement of the **Xbro***wser* display for this "Comparison" stage of a problem taken from a regular problem set generated for but not used in the experiment. Either the "Next Loop Slicer" or "Next Loop Annotater" button at the top of the left hand display is enabled according to ANNOTATION type: Manual or Annotated. Just below those buttons is an indicator (duplicated on the right side display) that shows whether the next problem concerned the first or second loop, with the inactive text blanked out. The control panel on the right side display is initially disabled so that subjects are required to acknowledge use of the appropriate tool – Slicer or Annotater – to proceed to the corresponding Manual or Annotated checklist. Thereafter returning to the Comparison Displays enables all but Quit; subjects were warned in advance that Quit would be disabled for the relatively short duration of a single problem.

The next phase of working a given problem involved the Problem Displays, arranged as in Fig. A.9, showing the last step of the first problem from the same example. The upper left display contains the current problem's loop slice, Annotated for this problem. The lower left display contains the Work Sheet of editors used with the checklist to determine the precise array elements affected by the loop transformation.. The lower right display contains the Answer Sheet of yes/no questions about the problem loop; text of questions remains concealed until the subject reaches Step ANS either by stating at the beginning of Step B that there are no loop-carried dependences by selecting a button, or by completely filling in the appropriate editors on the Work Sheet through Step D.3. This process is mediated by the checklist display in the upper right, containing one of twelve instructional displays for the steps of either the Manual or Annotated checklist as appropriate. In addition, a button beneath the checklist text returns to the Comparison displays, and the indicators next to it shows progress through the steps of the checklist, highlighting A, B1-B4, C, D1-D3 as each was completed and ANS once the Answer Sheet questions were all answered. Subjects then finished the problem by selecting "Register Answers/Continue" button on the ANS page of the checklist.

Figure A.8 Comparison Displays for first problem of problem set 5.3

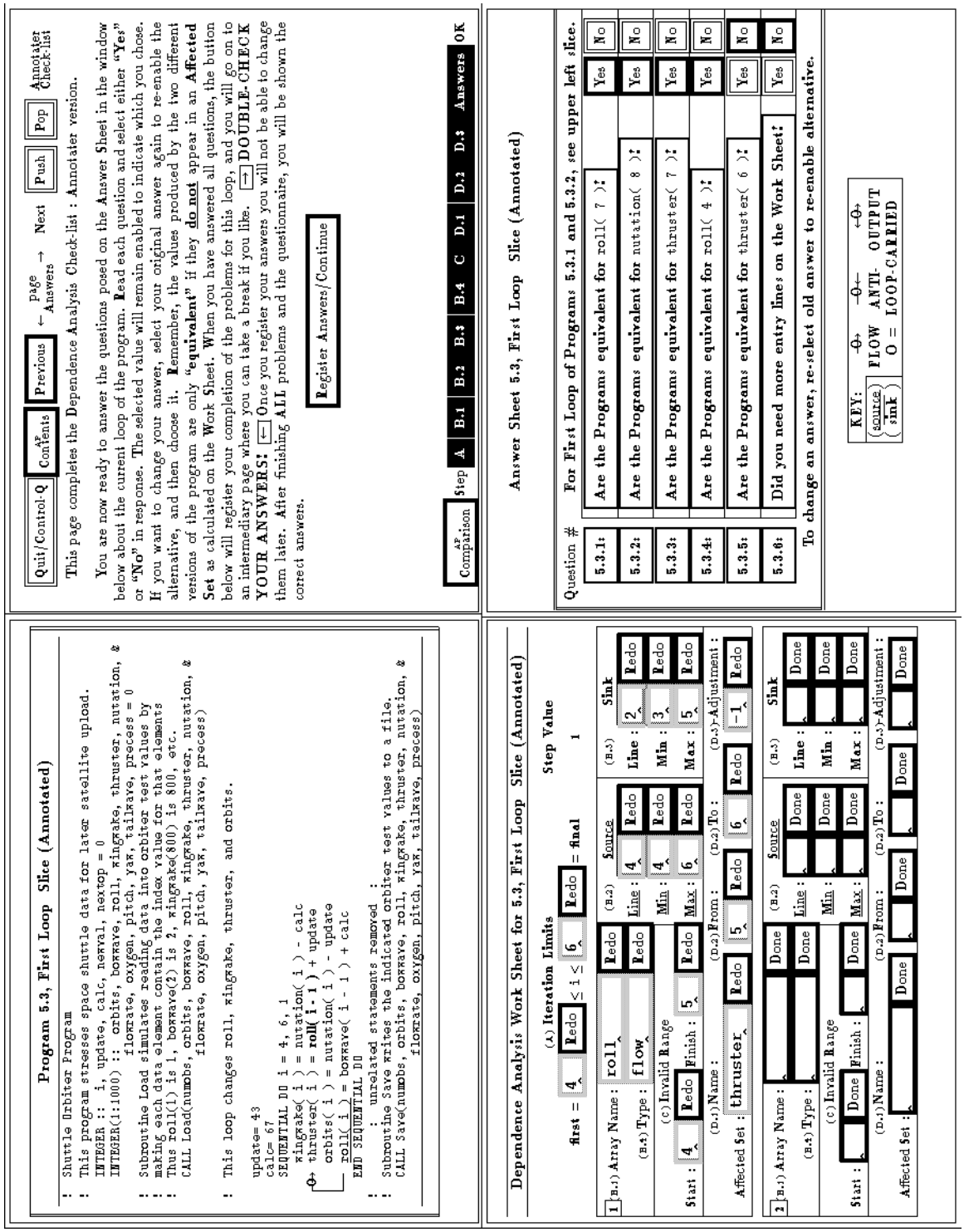


Figure A.9 Problem Displays for Step ANS
of first problem of problem set 5.3

Once a given problem was completed, subjects could take a break or continue. In the Review, each problem was followed by a twelve-display review of the problem similar to the Problem Displays with the checklist replaced with corresponding displays showing correct answers and requiring correction on the Work Sheet as needed before allowing the subject to continue.

The Review problem set contained one problem set of two problem loops each with two dependences: the first problem was Annotated, with a Reversed loop containing a Loop-Carried Anti- dependence and a Non-Loop-Carried Output dependence. The second problem was Manual, with a Parallel loop with a Loop-Carried Flow dependence and a Loop-Carried Anti- dependence. Including the four sets of checklist and review pages, the repeated Review problem set Comparison displays, and an intermediate "rest" display between problems, 60 displays in addition to the introductory material were required to complete the Review.

Section **5: P-F Program Problem Sets:** Pages 43-44 (578 displays)

Subjects now began the set of twenty-four randomly generated problems covering all combinations of the four within-subjects experimental factors. Each problem began with the Comparison displays for each problem set of two loops; thus there were twelve problem sets total with each loop's problem worked individually, requiring two Comparison displays, a Loop Slice or Annotation display, eighteen Checklist displays, a Work Sheet, an Answer Sheet, and a "rest" display between problems. With twenty-four displays for each problem, a subject viewed a total of 576 displays while working the problems in addition to the two-pages of introductory and concluding material for the problem section.

Section **6: Programming Background Questionnaire:** Pages 45-50

The background questionnaire consisted of 51 questions covering a variety of topics. Subjects were asked to answer by selecting from alternatives or entering text in response to specific questions.

Section **7: Debriefing:** Page 51 (plus optional material)

The debriefing explained that subjects had completed the study, and about the existence of the other TRAINING group. If desired, subjects could review their responses

and correct answers for the twenty-four regular problems, and could use **Xbro***wser* to learn the alternative method to that which they had learned for the study.

The following appendix contains additional details about the results of the experiment and the questionnaire, correlations tables, and a summary of subjects' performance experience using **Xbro***wser*.

Appendix B

Statistical Details

This appendix contains some of the statistical information too detailed to include in previous chapters. First, the complete results for the various analyses of variance (ANOVA) are shown for the error severity data for the experiment described in Chapter 7. Next, the ANOVA results for time to demonstrate comprehension in that same study are given. Then comes a section on the text and summary statistics for the interactive background questionnaire administered to the subjects of that experiment is presented. Next is a set of tables reporting some of the most interesting correlations of the questionnaire data with subjects' averages for ERROR severity and TASKTIME performance. Finally, general summary statistics about the subjects' participation profiles – such as time to complete the hypertext documents, or number of edit events – are given.

For both error severity and time to complete the problem checklist, the main technique employed was analysis of variance (ANOVA). The independent variables were CARRIED with two levels, ANNOTATION with two levels, DEPTYPE with three levels, and REVPAR with two levels all treated as within-subjects variables, and TRAINING with two levels treated as a between-subjects variable. Each subject worked 24 problems randomly covering all combinations of the within-subjects variables.

The F-statistic for each independent variable term and interaction terms combining them was constructed from the mean square variance in the appropriate dependent variable for each source term divided by the mean square variance for an appropriate denominator error term as suggested by Winer and McCall [Win71, McC86] The significance of that ratio was then tested against the F-distribution for the corresponding numerator and denominator degrees of freedom (DF). These statistics were calculated using version 6.09 of the GLM[1] procedure of SAS[2] [SAS90b].

---

[1]General Linear Modeling

[2]SAS is a trademark of the SAS Institute Inc.

B.1 Error Analyses

ANOVA were conducted for a dependent variable based on error severity scores ranging from 0 for no error occurring to 8 for total failure to comprehend a dependence. There were 16 subjects in the Algebraic Formulation TRAINING group, and 10 subjects in the Dependence Analysis TRAINING group. Three full analyses were conducted since the two levels of CARRIED, Non-Loop-Carried and Loop-Carried had very different distributions of scores: one analysis including both levels of CARRIED, and one for each level of CARRIED treated separately.

Abbreviations in the following tables are: S for SUBJECTS, T for TRAINING, C for CARRIED, A for ANNOTATION, D for DEPTYPE, and R for REVPAR. An asterisk appears between component abbreviations for each interaction term; S(T) indicates that TRAINING is nested for (*i.e.* is between-) SUBJECTS.

In addition to these main ANOVA, the relations between each pair of levels of DEPTYPE were tested by dropping observations for the unused level. Tables B.4 to B.1 present only these contrasts and corresponding error terms for the CARRIED-included, Non-Loop-Carried, and Loop-Carried analyses, with AVF an abbreviation for Anti- *vs.* Flow, AVO for Anti- *vs.* Output, and FVO for Flow *vs.* Output.

CARRIED-included ERROR analysis of variance

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
T	1	3.209	3.209	0.08875	0.76834
S(T)	24	867.702	36.154		
C	1	100.983	100.983	8.71555	0.00695
C*T	1	4.026	4.026	0.34747	0.56106
C*S(T)	24	278.077	11.587		
A	1	7.693	7.693	2.43889	0.13145
A*T	1	0.029	0.029	0.00926	0.92414
A*S(T)	24	75.702	3.154		
C*A	1	2.444	2.444	0.58664	0.45119
C*A*T	1	0.130	0.130	0.03125	0.86117
C*A*S(T)	24	99.972	4.166		
D	2	65.164	32.582	5.37485	0.00783
D*T	2	1.560	0.780	0.12869	0.87955
D*S(T)	48	290.972	6.062		
C*D	2	10.599	5.299	1.14633	0.32635
C*D*T	2	5.437	2.718	0.58800	0.55939
C*D*S(T)	48	221.903	4.623		
D*A	2	10.470	5.235	1.67575	0.19793
D*A*T	2	3.611	1.805	0.57794	0.56491
D*A*S(T)	48	149.947	3.124		
C*D*A	2	3.355	1.678	0.61985	0.54227
C*D*A*T	2	2.843	1.422	0.52519	0.59480
C*D*A*S(T)	48	129.920	2.707		
R	1	1.599	1.599	1.06892	0.31150
R*T	1	5.049	5.049	3.37607	0.07856
R*S(T)	24	35.893	1.496		
C*R	1	0.043	0.043	0.02919	0.86577
C*R*T	1	2.856	2.856	1.92901	0.17762
C*R*S(T)	24	35.535	1.481		
A*R	1	0.148	0.148	0.06567	0.79993
A*R*T	1	6.314	6.314	2.79384	0.10762
A*R*S(T)	24	54.243	2.260		
C*A*R	1	5.530	5.530	2.90926	0.10098
C*A*R*T	1	4.769	4.769	2.50857	0.12632
C*A*R*S(T)	24	45.622	1.901		
D*R	2	0.631	0.315	0.19228	0.82571
D*R*T	2	3.820	1.910	1.16503	0.32058
D*R*S(T)	48	78.699	1.640		
C*D*R	2	0.205	0.102	0.05973	0.94209
C*D*R*T	2	6.707	3.353	1.95435	0.15277
C*D*R*S(T)	48	82.364	1.716		
D*A*R	2	13.138	6.569	2.54956	0.08865
D*A*R*T	2	6.499	3.250	1.26121	0.29253
D*A*R*S(T)	48	123.674	2.577		
C*D*A*R	2	2.792	1.396	0.82547	0.44415
C*D*A*R*T	2	9.753	4.877	2.88399	0.06565
C*D*A*R*S(T)	48	81.164	1.691		

Table B.1 ANOVA for CARRIED-included error severity

Non-Loop-Carried ERROR analysis of variance

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
T	1	7.212	7.212	0.34889	0.56027
S(T)	24	496.083	20.670		
A	1	9.402	9.402	2.21930	0.14932
A*T	1	0.141	0.141	0.03336	0.85660
A*S(T)	24	101.679	4.237		
D	2	11.634	5.817	2.21881	0.11977
D*T	2	0.774	0.387	0.14754	0.86322
D*S(T)	48	125.835	2.622		
D*A	2	1.178	0.589	0.18331	0.83309
D*A*T	2	6.101	3.051	0.94909	0.39424
D*A*S(T)	48	154.290	3.214		
R	1	0.560	0.560	0.52597	0.47532
R*T	1	0.155	0.155	0.14572	0.70602
R*S(T)	24	25.550	1.065		
A*R	1	1.920	1.920	1.95176	0.17518
A*R*T	1	0.054	0.054	0.05506	0.81648
A*R*S(T)	24	23.613	0.984		
D*R	2	0.764	0.382	0.37654	0.68824
D*R*T	2	6.043	3.022	2.97866	0.06034
D*R*S(T)	48	48.694	1.014		
D*A*R	2	10.750	5.375	2.52311	0.09080
D*A*R*T	2	2.750	1.375	0.64547	0.52890
D*A*R*S(T)	48	102.256	2.130		

Table B.2 ANOVA for Non-Loop-Carried error severity

Loop-Carried ERROR analysis of variance

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
T	1	0.023	0.023	0.00086	0.97691
S(T)	24	649.695	27.071		
A	1	0.732	0.732	0.23743	0.63049
A*T	1	0.018	0.018	0.00585	0.93968
A*S(T)	24	73.995	3.083		
D	2	64.078	32.039	3.97345	0.02530
D*T	2	6.223	3.112	0.38590	0.68193
D*S(T)	48	387.040	8.063		
D*A	2	12.610	6.305	2.41007	0.10060
D*A*T	2	0.352	0.176	0.06735	0.93496
D*A*S(T)	48	125.577	2.616		
R	1	1.086	1.086	0.56833	0.45826
R*T	1	7.750	7.750	4.05426	0.05540
R*S(T)	24	45.878	1.912		
A*R	1	3.764	3.764	1.18459	0.28723
A*R*T	1	11.029	11.029	3.47126	0.07473
A*R*S(T)	24	76.253	3.177		
D*R	2	0.062	0.031	0.01315	0.98694
D*R*T	2	4.484	2.242	0.95766	0.39100
D*R*S(T)	48	112.369	2.341		
D*A*R	2	5.002	2.501	1.17029	0.31897
D*A*R*T	2	13.502	6.751	3.15896	0.05142
D*A*R*S(T)	48	102.581	2.137		

Table B.3 ANOVA for Loop-Carried error severity

CARRIED-included ERROR DEPTYPE Contrasts

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
AVF: Anti- *vs.* Flow	1	6.346	6.346	2.46038	0.12984
AVF*S(T)	24	61.902	2.579		
AVO: Anti- *vs.* Output	1	60.932	60.932	5.98000	0.02218
AVO*S(T)	24	244.542	10.189		
FVO: Flow *vs.* Output	1	28.968	28.968	3.54315	0.07197
FVO*S(T)	24	196.221	8.176		

Table B.4 DEPTYPE contrast ANOVA for CARRIED-included error severity

Non-Loop-Carried ERROR DEPTYPE Contrasts

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
AVF: Anti- *vs.* Flow	1	1.430	1.430	1.09167	0.30651
AVF*S(T)	24	31.442	1.310		
AVO: Anti- *vs.* Output	1	11.318	11.318	3.32109	0.08088
AVO*S(T)	24	81.792	3.408		
FVO: Flow *vs.* Output	1	4.702	4.702	1.49424	0.23343
FVO*S(T)	24	75.519	3.147		

Table B.5 DEPTYPE contrast ANOVA for Non-Loop-Carried error severity

Loop-Carried ERROR DEPTYPE Contrasts

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
AVF: Anti- *vs.* Flow	1	5.601	5.601	1.06261	0.31290
AVF*S(T)	24	126.513	5.271		
AVO: Anti- *vs.* Output	1	60.932	60.932	5.98000	0.02218
AVO*S(T)	24	244.542	10.189		
FVO: Flow *vs.* Output	1	29.584	29.584	3.38906	0.07803
FVO*S(T)	24	209.505	8.729		

Table B.6 DEPTYPE contrast ANOVA for Loop-Carried error severity

B.2 Time Analyses

ANOVA were conducted for a dependent variable based on the inverse of time required to complete a checklist demonstrating comprehension of the effects of a loop transformation. There were 12 subjects in the Algebraic Formulation TRAINING group, and 6 subjects in the Dependence Analysis TRAINING group. Since the times for Non-Loop-Carried and Loop-Carried problems were so different, only the two separate analyses were performed.

Abbreviations in the following tables are: S for SUBJECTS, T for TRAINING, A for ANNOTATION, D for DEPTYPE, and R for REVPAR. An asterisk appears between component abbreviations for each interaction term; S(T) indicates that TRAINING is nested for (*i.e.* is between-) SUBJECTS.

In addition to these main ANOVA, the relations between each pair of levels of DEPTYPE were tested by dropping observations for the unused level. Tables B.9 and B.10 present only these contrasts and corresponding error terms for the separate Non-Loop-Carried and Loop-Carried analyses, with AVF an abbreviation for Anti- *vs.* Flow, AVO for Anti- *vs.* Output, and FVO for Flow *vs.* Output.

Non-Loop-Carried TASKTIME analysis of variance

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
T	1	1.2550E-04	1.2550E-04	0.82168	0.37814
S(T)	16	2.4439E-03	1.5274E-04		
A	1	2.1097E-03	2.1097E-03	32.51162	0.00003
T*A	1	7.3102E-05	7.3102E-05	1.12652	0.30428
A*S(T)	16	1.0383E-03	6.4892E-05		
D	2	1.7786E-05	8.8928E-06	0.14440	0.86610
T*D	2	9.7754E-05	4.8877E-05	0.79365	0.46089
D*S(T)	32	1.9707E-03	6.1585E-05		
A*D	2	1.3058E-04	6.5291E-05	1.38632	0.26460
T*A*D	2	3.0159E-06	1.5079E-06	0.03202	0.96852
A*D*S(T)	32	1.5071E-03	4.7097E-05		
R	1	1.2363E-05	1.2363E-05	0.48951	0.49419
T*R	1	1.0513E-05	1.0513E-05	0.41626	0.52795
R*S(T)	16	4.0410E-04	2.5256E-05		
A*R	1	3.6027E-06	3.6027E-06	0.13697	0.71617
T*A*R	1	4.3466E-05	4.3466E-05	1.65247	0.21692
A*R*S(T)	16	4.2086E-04	2.6304E-05		
D*R	2	1.1928E-04	5.9639E-05	1.33661	0.27701
T*D*R	2	7.1501E-06	3.5751E-06	0.08012	0.92319
D*R*S(T)	32	1.4278E-03	4.4620E-05		
A*D*R	2	1.0492E-04	5.2461E-05	0.80802	0.45463
T*A*D*R	2	9.7096E-05	4.8548E-05	0.74776	0.48152
A*D*R*S(T)	32	2.0776E-03	6.4925E-05		

Table B.7 ANOVA for Non-Loop-Carried comprehension time

Loop-Carried TASKTIME analysis of variance

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
T	1	1.8280E-05	1.8280E-05	1.93122	0.18366
S(T)	16	1.5145E-04	9.4654E-06		
A	1	1.0293E-05	1.0293E-05	8.64842	0.00959
T*A	1	7.6043E-07	7.6043E-07	0.63890	0.43581
A*S(T)	16	1.9043E-05	1.1902E-06		
D	2	5.4422E-05	2.7211E-05	7.63348	0.00195
T*D	2	9.7166E-06	4.8583E-06	1.36291	0.27037
D*S(T)	32	1.1407E-04	3.5647E-06		
A*D	2	7.9889E-06	3.9945E-06	1.29969	0.28662
T*A*D	2	4.2985E-06	2.1493E-06	0.69931	0.50436
A*D*S(T)	32	9.8348E-05	3.0734E-06		
R	1	8.2478E-08	8.2478E-08	0.03255	0.85909
T*R	1	1.4279E-06	1.4279E-06	0.56350	0.46375
R*S(T)	16	4.0544E-05	2.5340E-06		
A*R	1	1.0624E-05	1.0624E-05	1.71628	0.20867
T*A*R	1	7.6203E-07	7.6203E-07	0.12311	0.73026
A*R*S(T)	16	9.9039E-05	6.1899E-06		
D*R	2	2.7665E-06	1.3832E-06	0.50291	0.60947
T*D*R	2	1.0138E-06	5.0692E-07	0.18431	0.83256
D*R*S(T)	32	8.8013E-05	2.7504E-06		
A*D*R	2	6.1641E-08	3.0820E-08	0.02011	0.98011
T*A*D*R	2	3.9489E-06	1.9745E-06	1.28810	0.28971
A*D*R*S(T)	32	4.9051E-05	1.5328E-06		

Table B.8 ANOVA for Loop-Carried comprehension time

Non-Loop-Carried TASKTIME DEPTYPE Contrasts

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
AVF: Anti- *vs.* Flow	1	8.3679E-06	8.3679E-06	0.20016	0.66059
AVF*S(T)	16	6.6890E-04	4.1806E-05		
AVO: Anti- *vs.* Output	1	1.4673E-06	1.4673E-06	0.01595	0.90107
AVO*S(T)	16	1.4718E-03	9.1989E-05		
FVO: Flow *vs.* Output	1	1.6843E-05	1.6843E-05	0.33051	0.57335
FVO*S(T)	16	8.1537E-04	5.0961E-05		

Table B.9 DEPTYPE contrast ANOVA for Non-Loop-Carried comprehension time

Loop-Carried TASKTIME DEPTYPE Contrasts

←Source or Error→	DF	Type III SS	Mean Square	*F*-ratio	p > *F*
AVF: Anti- *vs.* Flow	1	1.0004E-06	1.0004E-06	0.52093	0.48086
AVF*S(T)	16	3.0726E-05	1.9204E-06		
AVO: Anti- *vs.* Output	1	4.6647E-05	4.6647E-05	11.29673	0.00397
AVO*S(T)	16	6.6068E-05	4.1292E-06		
FVO: Flow *vs.* Output	1	3.3985E-05	3.3985E-05	7.31745	0.01561
FVO*S(T)	16	7.4310E-05	4.6444E-06		

Table B.10 DEPTYPE contrast ANOVA for Loop-Carried comprehension time

B.3 Subject Questionnaire and Analyses

After completing the regular problem sets, subjects used **Xbro***wser* to answer a background questionnaire of multiple choice and editor-entry questions covering topics considered possibly relevant to general or parallel programming ability as reported by Hammer [Ham84]. The questions provide two types of data about subjects: continuous responses – such as the number of years a subject had been programming – and ordinal responses on standardized Likert scales as described by Parten [Par50] – such as confidence in programming ability: Not at all/Somewhat/Fairly/Extremely. For continuous and ordinal responses Wilcoxon rank-sum tests (equivalent to the Mann-Whitney U statistic) were performed to compare location parameters for the TRAINING groups [GC92]; ordinal responses also were explored with regular χ^2 tests for general association and Mantel-Haenszel χ^2 tests for linear association [Fre87]. The complete text for each question below indicates a NAME to be used as an index to the numeric results and tabular data following. For simple continuous responses a table is given comparing the TRAINING groups separately and combined, with number of responses received (N), mean, standard deviation, minimum, median, and maximum values, together with the p probability of the associated Wilcoxon test. For most ordinal responses a table of RESPONSE×TRAINING group is presented with the response values (and associated ordinal value where applicable), row frequency count, cell frequency and associated column percent to allow comparison of the TRAINING groups, together with the p values for the associated χ^2 and Mantel-Haenszel tests; where appropriate the corresponding Wilcoxon test result is also reported.

Question 6.1 How many years have you been :
programming?† YEARS_PRG parallel programming?† YEARS_PAR

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
YEARS_PRG	AF:	16	8.3	3.6	2.0	9.5	14.0	
	DA:	10	8.0	3.1	2.0	9.5	11.0	
	BOTH:	26	8.2	3.3	2.0	9.5	14.0	0.83
YEARS_PAR	AF:	16	0.2	0.4	0.0	0.0	1.0	
	DA:	10	0.2	0.6	0.0	0.0	2.0	
	BOTH:	26	0.2	0.5	0.0	0.0	2.0	0.67

†continuous

Question 6.2 Do you know how to juggle?$^{\ddagger YN}$ JUGGLER
If so, how many items can you juggle?† JUGGLE_N
Have you ever juggled with another person?$^{\ddagger YN}$ JUGGLE_PAR

JUGGLER		χ^2 p = 0.34				Mantel-Haenszel p = 0.35			
No	16 =	AF	11	68.8%	+	DA	5	50.0%
Yes	10 =	AF	5	31.2%	+	DA	5	50.0%

JUGGLE_N		χ^2 p = 0.29				Mantel-Haenszel p = 0.16			
1	1 =	AF	0	0.0%	+	DA	1	20.0%
2	1 =	AF	0	0.0%	+	DA	1	20.0%
3	8 =	AF	5	100.0%	+	DA	3	60.0%

JUGGLE_PAR		χ^2 p = 0.29				Mantel-Haenszel p = 0.29			
No	23 =	AF	15	93.8%	+	DA	8	80.0%
Yes	3 =	AF	1	6.2%	+	DA	2	20.0%

Question 6.3 How confident are you in your ability to do sequential programming? $^{\ddagger L1}$ SEQ_CONFIDENCE
How confident are you in your ability to do parallel programming?$^{\ddagger L1}$ PAR_CONFIDENCE

SEQ_CONFIDENCE		χ^2 p = 0.39			Mantel-Haenszel p = 0.70				Wilcoxon p = 0.95
0	0 =	AF	0	0.0%	+	DA	0	0.0%
1	1 =	AF	0	0.0%	+	DA	1	10.0%
2	7 =	AF	5	31.2%	+	DA	2	20.0%
3	18 =	AF	11	68.8%	+	DA	7	70.0%

PAR_CONFIDENCE		χ^2 p = 0.47			Mantel-Haenszel p = 0.75				Wilcoxon p = 0.93
0	6 =	AF	4	25.0%	+	DA	2	20.0%
1	14 =	AF	8	50.0%	+	DA	6	60.0%
2	5 =	AF	4	25.0%	+	DA	1	10.0%
3	1 =	AF	0	0.0%	+	DA	1	10.0%

$^{\ddagger YN}$Choice of Yes/No
†theoretically continuous but N unreasonably small
$^{\ddagger L1}$Likert scale – Not at all/Somewhat/Fairly/Extremely, scored 0.0 to 3.0

Question 6.4 Try to recall your scores for any of these standardized tests, if taken:

What were your SAT (Scholastic Aptitude Test) scores?

Verbal:† SAT_VERBAL Mathematical:† SAT_MATH

What were your GRE (Graduate Record Examination) scores?

Verbal:† GRE_VERBAL Mathematical:† GRE_MATH Analytical:† GRE_ANALYTICAL

What were your TOEFL (Test of English as a Foreign Language) scores?

Oral:† TOEFL_ORAL Written:† TOEFL_WRITTEN

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
SAT_VERBAL	AF:	16	663	75.9	510	675	770	
	DA:	8	675	57.8	560	675	760	
	BOTH:	24	667	69.3	510	675	770	0.85
SAT_MATH	AF:	16	749	41.1	640	760	800	
	DA:	8	764	28.8	730	755	800	
	BOTH:	24	754	37.5	640	760	800	0.56
SAT_SUM	AF:	16	1412	88.0	1260	1415	1530	
	DA:	8	1439	65.1	1310	1445	1520	
	BOTH:	24	1421	80.7	1260	1435	1530	0.42

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
GRE_VERBAL	AF:	6	697	80.9	580	690	800	
	DA:	2	715	49.5	680	715	750	
	BOTH:	8	701	71.4	580	695	800	0.87
GRE_MATH	AF:	6	790	8.9	780	790	800	
	DA:	3	777	25.2	750	780	800	
	BOTH:	9	786	15.9	750	790	800	0.50
GRE_ANALYTICAL	AF:	6	770	33.5	730	775	800	
	DA:	3	710	79.4	650	680	800	
	BOTH:	9	750	56.3	650	750	800	0.28

† continuous

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
TOEFL_ORAL	AF:	2	645	21.2	630	645	660	
	DA:	2	602	40.3	573	602	630	
	BOTH:	4	623	36.4	573	630	660	0.41
TOEFL_WRITTEN	AF:	1	600	0	600	600	600	
	DA:	2	638	21.2	623	638	653	
	BOTH:	3	625	26.6	600	623	653	0.54

Question 6.5 How many **computer science** courses have you taken in high school and college? High School:† HS_CS_COURSES College:† CO_CS_COURSES

In your estimation, which is closest to your **grade point average** for them:‡G CS_GPA

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
HS_CS_COURSES	AF:	16	2.6	2.7	0.0	2.0	10.0	
	DA:	10	1.4	1.3	0.0	1.5	4.0	
	BOTH:	26	2.2	2.3	0.0	2.0	10.0	0.29
CO_CS_COURSES	AF:	16	7.6	3.3	3.0	7.0	15.0	
	DA:	10	8.7	4.3	6.0	7.0	20.0	
	BOTH:	26	8.0	3.7	3.0	7.0	20.0	0.54

CS_GPA χ^2 p = 0.51 Mantel-Haenszel p = 0.79 Wilcoxon p = 0.67

(2) C	1 =	AF	1	6.2%	+	DA	0	0.0%
(3) B	10 =	AF	5	31.2%	+	DA	5	50.0%
(4) A	15 =	AF	10	62.5%	+	DA	5	50.0%

†continuous

‡G Choice of A/B/C/D/F, scored 4.0 to 0.0

Question 6.6 How many **mathematics** courses have you taken in high school and college? High School[†] HS_MATH_COURSES College[†] CO_MATH_COURSES

In your estimation, which is closest to your **grade point average** for them[‡G] MATH_GPA

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
HS_MATH_COURSES	AF:	16	5.4	2.2	3.0	4.0	11.0	
	DA:	10	4.8	2.5	2.0	4.0	10.0	
	BOTH:	26	5.2	2.3	2.0	4.0	11.0	0.54
CO_MATH_COURSES	AF:	16	5.3	2.7	2.0	5.0	12.0	
	DA:	10	3.8	2.0	2.0	3.0	8.0	
	BOTH:	26	4.7	2.5	2.0	4.0	12.0	0.12

MATH_GPA χ^2 p = 0.22 Mantel-Haenszel p = 0.50 Wilcoxon p = 0.32

(2) C	1 =	AF	1	6.2%	+	DA	0	0.0%
(3) B	6 =	AF	2	12.5%	+	DA	4	40.0%
(4) A	19 =	AF	13	81.2%	+	DA	6	60.0%

Question 6.7 How familiar do you feel with Articulated Linkage for parallel programming? How confident are you in your ability to use Articulated Linkage? Where did you first learn about Articulated Linkage?[1]

Question 6.8 How familiar do you feel with *method*[2] for parallel programming?[‡L1] METHOD_FAMILIAR

How confident are you in your ability to use *method*?[‡L1] METHOD_CONFIDENCE

Where did you first learn about *method*?[3] METHOD_LEARNED

METHOD_FAMILIAR χ^2 p = 0.51 Mantel-Haenszel p = 0.90 Wilcoxon p = 0.73

0	2 =	AF	1	6.2%	+	DA	1	11.1%
1	17 =	AF	12	75.0%	+	DA	5	55.6%

[†]continuous

[‡G]Choice of A/B/C/D/F, scored 4.0 to 0.0

[1]No responses given about this *nonexistent* TRAINING method used to check reply accuracy.

[2]Where *method* is the TRAINING group method learned in the study.

[‡L1]Likert scale – Not at all/Somewhat/Fairly/Extremely, scored 0.0 to 3.0

[3]Open responses, summarized as "here"=This study, or "else"=elsewhere

2	5 =	AF	2	12.5%	+	DA	3	33.3%
3	1 =	AF	1	6.2%	+	DA	0	0.0%

METHOD_CONFIDENCE		χ^2 p = 0.39			Mantel-Haenszel p = 0.62			Wilcoxon p = 0.76
0	3 =	AF	1	6.2%	+	DA	2	20.0%
1	19 =	AF	13	81.2%	+	DA	6	60.0%
2	3 =	AF	1	6.2%	+	DA	2	20.0%
3	1 =	AF	1	6.2%	+	DA	0	0.0%

METHOD_LEARNED				χ^2 p = 0.17				Mantel-Haenszel p = 0.18
else	1 =	AF	0	0.0%	+	DA	1	11.1%
here	24 =	AF	16	100.0%	+	DA	8	88.9%

Question 6.9 How familiar do you feel with *alternate*[4] for parallel programming?$^{\ddagger L1}$ ALT_FAMILIAR

How confident are you in your ability to use *alternate*?$^{\ddagger L1}$ ALT_CONFIDENCE

Where did you first learn about *alternate*?[5] ALT_LEARNED

ALT_FAMILIAR				χ^2 p = -				Mantel-Haenszel p = -
1	3 =	AF	3	75.0%	+	DA	-	-%
2	1 =	AF	1	25.0%	+	DA	-	-%

ALT_CONFIDENCE				χ^2 p = -				Mantel-Haenszel p = -
0	1 =	AF	1	25.0%	+	DA	-	-%
1	2 =	AF	2	50.0%	+	DA	-	-%
2	1 =	AF	1	25.0%	+	DA	-	-%

Question 6.10 Have you ever managed or controlled a group of people in which each member was doing something different at the same time?$^{\ddagger YN}$ MANAGER

If so, how large was the largest such group you handled?† NUM_MANAGED

[4]Where *alternate* is the TRAINING group method **not** learned in the study.

$^{\ddagger L1}$Likert scale – Not at all/Somewhat/Fairly/Extremely, scored 0.0 to 3.0

[5]Open responses, but all categorized as Elsewhere

$^{\ddagger YN}$Choice of Yes/No

†continuous

MANAGER		χ^2 p = 0.90				Mantel-Haenszel p = 0.90			
No	10 =	AF	6	37.5%	+	DA	4	40.0%	
Yes	16 =	AF	10	62.5%	+	DA	6	60.0%	

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
NUM_MANAGED	AF:	10	7.2	7.3	2.0	4.0	25.0	
	DA:	6	7.0	7.0	2.0	3.5	20.0	
	BOTH:	16	7.1	6.9	2.0	4.0	25.0	0.70

Question 6.11 How many **informal** courses relevant to either sequential or parallel programming have you taken? Include here any non-credit courses, seminars, workshops, *etc.*:† INFORMAL_COURSES

Compared to other learning methods, how useful were these?‡^{L1}

INFORMAL_VALUE

How many of these courses covered parallel programming? Compared to other learning methods, how useful were these?[6]

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
INFORMAL_COURSES	AF:	16	0.5	1.2	0.0	0.0	4.0	
	DA:	10	0.7	1.1	0.0	0.0	3.0	
	BOTH:	26	0.6	1.1	0.0	0.0	4.0	0.36
INFORMAL_COURSES> 0[7]	AF:	3	2.7	1.2	2.0	2.0	4.0	
	DA:	4	1.8	1.0	1.0	1.5	3.0	
	BOTH:	7	2.1	1.1	1.0	2.0	4.0	0.35

INFORMAL_VALUE		χ^2 p = 0.14				Mantel-Haenszel p = 0.09			Wilcoxon p = 0.11
0	2 =	AF	0	0.0%	+	DA	2	50.0%	
1	1 =	AF	0	0.0%	+	DA	1	25.0%	
2	4 =	AF	3	100.0%	+	DA	1	25.0%	

HAD_TAKEN		χ^2 p = 0.23				Mantel-Haenszel p = 0.24			
n= 0	19 =	AF	13	81.2%	+	DA	6	60.0%	

†continuous

‡^{L1}Likert scale – Not at all/Somewhat/Fairly/Extremely, scored 0.0 to 3.0

[6]One subject in the Algebraic Formulation group had taken one informal course and ranked it "Somewhat" valuable.

[7]statistics for numbers > 0 only

n> 0	7 =	AF	3	18.8%	+	DA	4	40.0%

Question 6.12 Compared to formal and informal instruction, how useful have you found any other experiences you think contributed significantly to your general programming ability, such as books or self-taught methods?‡L1 OTHER_VALUE

How useful have any been to your parallel programming ability?‡L1 OTHER_PAR_VALUE

OTHER_VALUE		χ^2 p = 0.23			Mantel-Haenszel p = 0.10				Wilcoxon p = 0.11
1	6 =	AF	2	13.3%	+	DA	4	40.0%
2	7 =	AF	4	26.7%	+	DA	3	30.0%
3	12 =	AF	9	60.0%	+	DA	3	30.0%

OTHER_PAR_VALUE		χ^2 p = 0.39			Mantel-Haenszel p = 0.16				Wilcoxon p = 0.17
0	2 =	AF	1	12.5%	+	DA	1	25.0%
1	6 =	AF	3	37.5%	+	DA	3	75.0%
2	3 =	AF	3	37.5%	+	DA	0	0.0%
3	1 =	AF	1	12.5%	+	DA	0	0.0%

Question 6.13 What percentage of working time do you spend doing:

non-programming related activities?† NON_PROG_PCT

sequential programming?† SEQ_PROG_PCT

parallel programming?[8]

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
NON_PROG_PCT	AF:	15	53.1	22.5	20.0	50.0	99.0	
	DA:	10	42.5	25.3	10.0	40.0	80.0	
	BOTH:	25	48.8	23.8	10.0	50.0	99.0	0.32
SEQ_PROG_PCT	AF:	15	46.9	22.5	1.0	50.0	80.0	
	DA:	10	53.5	24.5	20.0	50.0	90.0	
	BOTH:	25	49.6	23.1	1.0	50.0	90.0	0.60

‡L1 Likert scale – Not at all/Somewhat/Fairly/Extremely, scored 0.0 to 3.0

† continuous

[8] One subject in the Dependence Analysis group responded 40% for parallel programming.

Question 6.14 How many programming languages do you know? List as many as you can recall. If you have used it within the last six months, put an asterisk - "*" - after it. If it is a parallel language, put it in (parentheses.) [9] LANGS_TOTAL

Most languages appear separately here if used more than once, or as "other" for non-parallel and "(parallel)" for parallel languages. Full credit is given for multiple dialects. "other" languages include Action!, Ada, COBOL, DBase, Eiffel, forth, G/Labview II, G2, PL/SQL, RCL, SPICE, and SQL*Plus. "(parallel)" languages include ANL macros/PARMACS, C*, CSP (Hoare's Communicating Sequential Processes), iPSC/2 C, PARCSIM, and PCN from CalTech, and were reported by only one subject in each TRAINING group. "utility/shell" languages included awk, perl, csh, sh, *etc.* Each number pair listed is <LANGS_TOTAL, LANGS_6MOS*>.

assembly	=	AF	8,1*	6,2*%	+	DA	14,3*	16,8*%
BASIC	=	AF	18,3*	13,5*%	+	DA	8,1*	9,3*%
C	=	AF	14,10*	10,16*%	+	DA	10,9*	11,24*%
C++	=	AF	14,9*	10,15*%	+	DA	9,8*	10,21*%
FORTRAN	=	AF	10,4*	7,6*%	+	DA	8,1*	9,3*%
LISP/Scheme	=	AF	19,15*	14,24*%	+	DA	14,8*	16,21*%
Logo	=	AF	3,0*	2,0*%	+	DA	4,0*	4,0*%
Maple/Matlab	=	AF	9,5*	7,8*%	+	DA	0,0*	0,0*%
ML	=	AF	3,3*	2,5*%	+	DA	0,0*	0,0*%
Modula-2	=	AF	1,0*	1,0*%	+	DA	1,0*	1,0*%
Pascal	=	AF	14,1*	10,2*%	+	DA	10,2*	11,5*%
PostScript	=	AF	1,0*	1,0*%	+	DA	1,1*	1,3*%
Prolog	=	AF	6,0*	4,0*%	+	DA	2,0*	2,0*%
"other"	=	AF	9,4*	7,6*%	+	DA	3,2*	3,5*%
"(parallel)"	=	AF	3,3*	2,5*%	+	DA	3,1*	3,3*%
"utility/shell"	=	AF	6,4*	4,6*%	+	DA	2,2*	2,5*%
TOTALS	=	AF	138,62*	100,100*%	+	DA	89,38*	98,101*%

[9] For each subject, LANGS_TOTAL is the total reported, and LANGS_6MOS the sum of those used within last six months, with full credit given for each dialect.

*when restricted to languages used in last six months

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
LANGS_6MOS	AF:	16	3.9	2.7	0.0	3.5	10.0	
	DA:	10	3.7	1.8	2.0	3.0	8.0	
	BOTH:	26	3.8	2.4	0.0	3.0	10.0	0.98
LANGS_TOTAL	AF:	16	8.6	3.1	5.0	8.5	15.0	
	DA:	10	8.2	3.9	5.0	6.0	17.0	
	BOTH:	26	8.5	3.3	5.0	7.5	17.0	0.63

Question 6.15 How valuable in parallel programming do you think you would find the Manual method of *method*[10] you learned in this study?‡^{L1} MANUAL_VALUE

MANUAL_VALUE χ^2 p = 0.54 Mantel-Haenszel p = 0.83 Wilcoxon p = 0.91

0	1 =	AF	1	6.2%	+	DA	0	0.0%
1	12 =	AF	7	43.8%	+	DA	5	50.0%
2	11 =	AF	6	37.5%	+	DA	5	50.0%
3	2 =	AF	2	12.5%	+	DA	0	0.0%

Question 6.16 Assuming the Annotater was always available, how valuable in parallel programming do you think you would find the Annotater method of *method*[10] you learned in this study?‡^{L1} ANNOTATED_VALUE

ANNOTATED_VALUE χ^2 p = 0.52 Mantel-Haenszel p = 0.62 Wilcoxon p = 0.77

1	3 =	AF	1	6.2%	+	DA	2	20.0%
2	10 =	AF	7	43.8%	+	DA	3	30.0%
3	13 =	AF	8	50.0%	+	DA	5	50.0%

[10]Where *method* is the TRAINING group method learned in the study.

‡^{L1}Likert scale – Not at all/Somewhat/Fairly/Extremely, scored 0.0 to 3.0

Question 6.17 Did you enjoy working the problems (as opposed to learning the method)?‡L1 ENJOYMENT

ENJOYMENT		χ^2 p = 0.16			Mantel-Haenszel p = 0.74			Wilcoxon p = 0.91
0	5 =	AF	4	25.0%	+	DA	1	10.0%
1	8 =	AF	3	18.8%	+	DA	5	50.0%
2	10 =	AF	8	50.0%	+	DA	2	20.0%
3	3 =	AF	1	6.2%	+	DA	2	20.0%

Question 6.18 Indicate how frequently you use the following programming language features:‡L2

`IF` statements:	IF_FREQ
sequential `DO` loop statements:	SEQ_DO_FREQ
`PARALLEL DO` loop statements:	PAR_DO_FREQ
sequential `SELECT`/`switch`/`CASE` statements:	SEQ_CASE_FREQ
`PARALLEL REGION`/`SELECT`/`switch`/`CASE` statements:	PAR_CASE_FREQ
recursion:	RECURSION_FREQ
pointer variables:	POINTER_FREQ

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
IF_FREQ	AF:	16	3.6	0.6	2.0	4.0	4.0	
	DA:	10	3.7	0.5	3.0	4.0	4.0	
	BOTH:	26	3.6	0.6	2.0	4.0	4.0	0.66
SEQ_DO_FREQ	AF:	16	2.7	1.2	0.0	3.0	4.0	
	DA:	10	3.4	0.7	2.0	3.5	4.0	
	BOTH:	26	3.0	1.1	0.0	3.0	4.0	0.16
PAR_DO_FREQ	AF:	16	0.0	0.0	0.0	0.0	0.0	
	DA:	10	0.0	0.0	0.0	0.0	0.0	
	BOTH:	26	0.0	0.0	0.0	0.0	0.0	1.00
SEQ_CASE_FREQ	AF:	16	2.4	0.9	1.0	2.0	4.0	
	DA:	10	2.1	1.2	0.0	2.0	4.0	
	BOTH:	26	2.3	1.0	0.0	2.0	4.0	0.53

‡L2 Likert scale – Never/Occasionally/Moderately/Frequently/Constantly, scored 0.0 to 4.0 – only used numerically here except for RECURSION_FREQ

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
PAR_CASE_FREQ	AF:	16	0.0	0.0	0.0	0.0	0.0	
	DA:	10	0.0	0.0	0.0	0.0	0.0	
	BOTH:	26	0.0	0.0	0.0	0.0	0.0	1.00
RECURSION_FREQ	AF:	16	3.1	0.7	2.0	3.0	4.0	
	DA:	10	2.4	0.7	1.0	2.5	3.0	
	BOTH:	26	2.8	0.7	1.0	3.0	4.0	0.04

RECURSION_FREQ χ^2 p = 0.16 Mantel-Haenszel p = 0.028 Wilcoxon p = 0.04

1	1 =	AF	0	0.0%	+	DA	1	10.0%
2	7 =	AF	3	18.8%	+	DA	4	40.0%
3	14 =	AF	9	56.2%	+	DA	5	50.0%
4	4 =	AF	4	25.0%	+	DA	0	0.0%

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
POINTER_FREQ	AF:	16	3.1	1.0	1.0	3.0	4.0	
	DA:	10	3.4	0.7	2.0	3.5	4.0	
	BOTH:	26	3.2	0.9	1.0	3.0	4.0	0.46

B.4 Selected Questionnaire and Performance Correlation Data

A *post hoc* analysis was performed using the questionnaire data collected, examining possible correlations with subjects' performance on error severity and time to demonstrate loop transformation comprehension as described in Chapter 7. Only correlations with $p < 0.05$ are reported there. Additional correlations of smaller magnitude and significance or that did not fit a reasonable overall picture are presented here. Results are given in the following tables for Pearson product-moment correlations of subjects' averages on ERROR severity and TASKTIME comprehension measures controlling for the effects of CARRIED, ANNOTATION, and DEPTYPE, contrasted with data from the questionnaire described in Chapter B.3. Each cell of the Correlation Tables contain the Pearson R correlation, the probability of a greater R occurring, and the number of observations involved. To be included in these tables, the data involved had to be involved with at least one correlation with $p < 0.10$: such correlations are highlighted in the table with **bold** text and a heavy outline.

To provide context for these correlations, the following two tables present some summary statistics for the questionnaire data and subject averages used in the correlations, respectively.

The Questionnaire table gives question number references, a summary of the corresponding meaning, a variable name used as an index in the correlations tables, and the summary statistics.

The Subject Averages table divides the summary statistics for subjects' averages into several subtables, contrasting Non-Loop-Carried and Loop-Carried (levels of CARRIED) with ERROR and TASKTIME results. The various statistics represent an overall average within each section, and averages for Annotated and Manual (levels of ANNOTATED) problems, Anti-, Flow, and Output dependences (levels of DEPTYPE) problems, and their various combinations. Note that all subjects were able to contribute TASKTIME values for their personal averages: individual observations resulting from ERROR severity greater than 6 were excluded, not the entire data set for such subjects as in the time ANOVA.

The Correlations tables following thereafter are in order: ERROR Non-Loop-Carried, ERROR Loop-Carried, TASKTIME Non-Loop-Carried, and TASKTIME Loop-Carried. Each table is five pages long, and covers each combination of the questionnaire data and subject averages summarized below.

Summary Statistics of Questionnaire Data Reported in Correlations Tables							
Question #	Meaning	Variable=	N	Mean	SD	Min	Max
6.01a	# of years of programming experience	QPROGYRS	26	8.19	3.33	2	14
6.02a	Able to Juggle	QVJUGGLE	26	0.38	0.50	0	1
6.03a	Confidence in Sequential Programming	QCNFSQPG	26	2.65	0.56	1	3
6.03b	Confidence in Parallel Programming	QCNFPRPG	26	1.04	0.77	0	3
6.04aa	SAT Verbal	QSATV	24	667.08	69.31	510	770
6.04ab	SAT Mathematics	QSATM	24	753.75	37.51	640	800
6.04a*/2	SAT Average	QSATAVG	24	710.42	40.35	630	765
6.04ba	GRE Verbal	QGREV	8	701.25	71.40	580	800
6.04bb	GRE Mathematics	QGREM	9	785.56	15.90	750	800
6.04bc	GRE Analytical	QGREA	9	750.00	56.35	650	800
6.04b*/3	GRE Average	QGREAVG	9	749.26	41.09	693	800
6.05a	# of High School Comp. Sci. courses	QHSCS	26	2.15	2.33	0	10
6.05b	# of College Comp. Sci. Courses	QCOCS	26	8.04	3.66	3	20
6.05a+b	# of H.S. + College Comp. Sci. courses	QCSSUM	26	10.19	4.36	5	20
6.05c	Comp. Sci. GPA	QCSGPA	26	3.54	0.58	2	4
6.06a	# of High School Mathematics Courses	QHSMATH	26	5.15	2.29	2	11
6.06b	# of College Mathematics Courses	QCOMATH	26	4.73	2.51	2	12
6.06a+b	# of H.S. + College Mathematics courses	QMATHSUM	26	9.88	3.95	4	18
6.06c	Mathematics GPA	QMATHGPA	26	3.69	0.55	2	4
6.08a	Familiarity of Method Learned	QMETHFAM	25	1.20	0.65	0	3
6.08b	Confidence in Method Learned	QMETHCNF	26	1.08	0.63	0	3
6.11a	# of informal courses in programming	QNINFRML	26	0.58	1.10	0	4
6.11b	Perceived Value of Informal Learning	QVINFRML	7	1.29	0.95	0	2
6.12a	Perceived usefulness of "other" experiences in programming	QGUSEFUL	25	2.24	0.83	1	3
6.13a	Percentage of Time in Non-Programming Work	QNPGPCT	25	48.84	23.76	10	99
6.13b	Percentage of Time in Seq. Programming Work	QSPGPCT	25	49.56	23.08	1	90
6.14a	# of Programming Languages used at any time	QALLLANG	26	8.35	3.35	5	16
6.14b	# of Programming Languages used within last 6 months	QACTLANG	25	4.08	2.08	2	10
6.15a	Perceived Value of Manual Method	QVMANMTH	26	1.54	0.71	0	3
6.16a	Perceived Value of Annotated Method	QVANNMTH	26	2.38	0.70	1	3
6.17a	Enjoyment of working problems	QENJOYED	26	1.42	0.95	0	3
6.18a	Frequency of use of: IF statements	QIFF	26	3.62	0.57	2	4
6.18b	Frequency of use of: Sequential DO	QSEQDOF	26	2.96	1.11	0	4
6.18d	Frequency of use of: CASE statement	QCASEF	26	2.31	1.01	0	4
6.18f	Frequency of use of: RECURSION	QRECURSE	26	2.81	0.75	1	4

Table B.11 Summary Statistics for Questionnaire Data Reported in Correlations Tables

Summary Statistics of Subject Averages used for Correlations Tables												
Non-Loop-Carried						Loop-Carried						
ERROR	N	Mean	SD	Min	Max	ERROR	N	Mean	SD	Min	Max
Overall	26	0.51	1.27	0	6.00	Overall	26	1.31	1.47	0	5.08
Annotated	26	0.35	1.08	0	5.33	Annotated	26	1.26	1.56	0	4.67
Manual	26	0.68	1.64	0	6.67	Manual	26	1.37	1.55	0	6.00
Anti- dependence	26	0.73	1.58	0	6.00	Anti- dependence	26	1.81	2.29	0	6.50
Flow dependence	26	0.52	1.65	0	8.00	Flow dependence	26	1.49	2.13	0	6.75
Output dependence	26	0.28	0.87	0	4.00	Output dependence	26	0.64	0.83	0	2.25
Anti- dep. Annotated	26	0.65	1.85	0	8.00	Anti- dep. Annotated	26	2.06	2.51	0	7.00
Anti- dep. Manual	26	0.81	1.95	0	8.00	Anti- dep. Manual	26	1.56	2.36	0	7.00
Flow dep. Annotated	26	0.35	1.57	0	8.00	Flow dep. Annotated	26	1.27	2.18	0	6.50
Flow dep. Manual	26	0.79	2.26	0	8.00	Flow dep. Manual	26	1.71	2.28	0	7.00
Output dep. Annotated	26	0.06	0.22	0	1.00	Output dep. Annotated	26	0.46	0.86	0	3.00
Output dep. Manual	26	0.50	1.73	0	8.00	Output dep. Manual	26	0.83	1.52	0	4.50
Summary Statistics of Subject Averages used for Correlations Tables												
Non-Loop-Carried						Loop-Carried						
TASKTIME	N	Mean	SD	Min	Max	TASKTIME	N	Mean	SD	Min	Max
Overall	26	55.9	14.3	38.5	92.5	Overall	26	191.4	29.7	139.9	245.7
Annotated	26	47.9	11.1	30.8	71.3	Annotated	26	183.1	33.6	124.8	246.0
Manual	26	64.0	22.8	40.2	136.5	Manual	26	198.9	32.5	120.0	266.4
Anti- dependence	26	55.4	22.3	33.5	118.0	Anti- dependence	26	208.6	40.5	131.5	289.8
Flow dependence	25	56.6	22.8	31.0	143.8	Flow dependence	26	199.9	38.6	149.0	308.5
Output dependence	26	55.0	15.5	33.0	92.0	Output dependence	26	165.6	31.6	104.8	229.5
Anti- dep. Annotated	25	54.0	32.5	30.5	180.0	Anti- dep. Annotated	25	199.8	53.7	120.0	345.0
Anti- dep. Manual	25	59.8	30.6	28.0	156.5	Anti- dep. Manual	25	217.2	38.6	143.0	282.5
Flow dep. Annotated	25	44.8	14.0	30.0	79.5	Flow dep. Annotated	26	182.6	50.9	113.5	381.5
Flow dep. Manual	24	69.6	42.1	37.5	238.5	Flow dep. Manual	25	219.6	47.5	156.0	321.5
Output dep. Annotated	26	47.1	16.5	20.5	83.5	Output dep. Annotated	26	165.2	36.8	112.5	253.0
Output dep. Manual	25	63.4	25.6	36.0	138.0	Output dep. Manual	26	165.9	34.6	80.5	233.5

Table B.12 Summary Statistics for ERROR and TASKTIME Reported in Correlations Tables

ERROR							
Non-Loop-Carried							
Question:	6.01a	6.02a	6.03a	6.03b	6.04aa	6.04ab	6.04a*/2
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	# of years of programming experience	Able to Juggle	Confidence in Sequential Programming	Confidence in Parallel Programming	SAT Verbal	SAT Mathematics	SAT Average
Variable=	QPROGYRS	QVJUGGLE	QCNFSQPG	QCNFPRPG	QSATV	QSATM	QSATAVG
Overall	0.09388	0.03365	0.14884	-0.07854	0.10691	**-0.68037**	-0.22443
Average	0.6483	0.8704	0.468	0.7029	0.619	**0.0003**	0.2917
	26	26	26	26	24	**24**	24
Annotated	0.01742	-0.10131	0.00042	-0.12044	0.17423	**-0.61556**	-0.13648
Average	0.9327	0.6224	0.9984	0.5578	0.4155	**0.0014**	0.5248
	26	26	26	26	24	**24**	24
Manual	0.13958	0.13227	0.23666	-0.0425	0.05054	**-0.6497**	-0.25858
Average	0.4965	0.5195	0.2444	0.8367	0.8146	**0.0006**	0.2224
	26	26	26	26	24	**24**	24
Anti-	0.11631	0.16263	0.09347	-0.05655	0.04429	**-0.54687**	-0.21616
dependence	0.5715	0.4273	0.6497	0.7838	0.8372	**0.0057**	0.3104
Average	26	26	26	26	24	**24**	24
Flow	0.14508	0.00031	0.18925	-0.02395	0.19332	**-0.68319**	-0.15152
dependence	0.4795	0.9988	0.3545	0.9075	0.3654	**0.0002**	0.4797
Average	26	26	26	26	24	**24**	24
Output	-0.07414	-0.14212	0.12321	-0.19426	0.01811	**-0.67458**	-0.29801
dependence	0.7189	0.4886	0.5488	0.3416	0.9331	**0.0003**	0.1573
Average	26	26	26	26	24	**24**	24
Anti-dep.	-0.04069	-0.08893	-0.1008	-0.15819	0.16681	**-0.53455**	-0.1052
Annotated	0.8436	0.6657	0.6242	0.4402	0.4359	**0.0071**	0.6247
Average	26	26	26	26	24	**24**	24
Anti-dep.	0.22678	**0.34738**	0.24664	0.05799	-0.08643	**-0.37744**	-0.24968
Manual	0.2652	**0.0821**	0.2245	0.7784	0.688	**0.069**	0.2393
Average	26	**26**	26	26	24	**24**	24
Flow dep.	0.08626	-0.15238	0.14161	-0.02792	0.16979	**-0.64612**	-0.1545
Annotated	0.6752	0.4574	0.4902	0.8923	0.4277	**0.0006**	0.471
Average	26	26	26	26	24	**24**	24
Flow dep.	0.17782	0.16439	0.20756	-0.01801	0.16696	**-0.56876**	-0.12097
Manual	0.3848	0.4223	0.3089	0.9304	0.4355	**0.0037**	0.5734
Average	26	26	26	26	24	**24**	24
Output dep.	-0.01604	**0.34498**	-0.15873	-0.25351	-0.04371	0.02262	-0.02703
Annotated	0.938	**0.0844**	0.4386	0.2114	0.8393	0.9165	0.9002
Average	26	**26**	26	26	24	24	24
Output dep.	-0.07296	-0.18681	0.14441	-0.16474	0.02376	**-0.68435**	-0.2977
Manual	0.7232	0.3608	0.4815	0.4213	0.9123	**0.0002**	0.1577
Average	26	26	26	26	24	**24**	24

Table B.13 Selected Correlations of Non-Loop-Carried ERROR with Questionnaire Data

ERROR							
Non-Loop-Carried							
Question:	6.04ba	6.04bb	6.04bc	6.04b*/3	6.05a	6.05b	6.05a+b
Table Cells Key: Pearson Correlation R Prob > R N obs.	GRE Verbal	GRE Mathematics	GRE Analytical	GRE Average	# of High School Comp. Sci. courses	# of College Comp. Sci. Courses	# of H.S. + College Comp. Sci. courses
Variable=	QGREV	QGREM	QGREA	QGREAVG	QHSCS	QCOCS	QCSSUM
Overall	-0.33547	0.10143	-0.0056	-0.2136	-0.15987	-0.1614	-0.22065
Average	0.4166	0.7951	0.9886	0.5811	0.4353	0.4309	0.2787
	8	9	9	9	26	26	26
Annotated	-0.34434	0.13924	0.03289	-0.2034	-0.15751	-0.0844	-0.15479
Average	0.4036	0.7209	0.9331	0.5997	0.4422	0.6819	0.4502
	8	9	9	9	26	26	26
Manual	-0.3262	0.08073	-0.0251	-0.2157	-0.14404	-0.19997	-0.24456
Average	0.4304	0.8364	0.9489	0.5773	0.4827	0.3273	0.2286
	8	9	9	9	26	26	26
Anti-	-0.31531	0.08915	-0.01677	-0.20314	-0.16208	-0.25708	-0.30208
dependence	0.4468	0.8196	0.9658	0.6001	0.4289	0.2049	0.1337
Average	8	9	9	9	26	26	26
Flow	-0.28673	0.14858	0.04192	-0.15331	-0.15917	-0.10473	-0.17273
dependence	0.4911	0.7028	0.9147	0.6937	0.4374	0.6106	0.3988
Average	8	9	9	9	26	26	26
Output	-0.1652	-0.04586	-0.04991	-0.15887	-0.10075	-0.04106	-0.08817
dependence	0.6958	0.9067	0.8985	0.6831	0.6243	0.8422	0.6684
Average	8	9	9	9	26	26	26
Anti-dep.	-0.31531	0.08915	-0.01677	-0.20314	-0.19174	-0.17239	-0.24686
Annotated	0.4468	0.8196	0.9658	0.6001	0.3481	0.3997	0.2241
Average	8	9	9	9	26	26	26
Anti-dep.	-0.31531	0.08915	-0.01677	-0.20314	-0.08116	-0.25327	-0.25574
Manual	0.4468	0.8196	0.9658	0.6001	0.6935	0.2119	0.2073
Average	8	9	9	9	26	26	26
Flow dep.	0.04952	0.34069	0.33276	0.18929	-0.17425	0.00107	-0.09202
Annotated	0.9073	0.3696	0.3816	0.6257	0.3946	0.9958	0.6548
Average	8	9	9	9	26	26	26
Flow dep.	-0.29002	0.10483	0	-0.17577	-0.11508	-0.18002	-0.21238
Manual	0.4859	0.7884	1	0.651	0.5756	0.3789	0.2976
Average	8	9	9	9	26	26	26
Output dep.	-0.14454	0.03706	0.03137	-0.10517	**0.53936**	0.1997	**0.45515**
Annotated	0.7327	0.9246	0.9361	0.7877	**0.0045**	0.328	**0.0195**
Average	8	9	9	9	**26**	26	**26**
Output dep.	-0.17684	-0.13104	-0.1331	-0.20619	-0.16927	-0.06647	-0.14603
Manual	0.6753	0.7368	0.7328	0.5945	0.4084	0.747	0.4766
Average	8	9	9	9	26	26	26

(Table B.13 continued)

ERROR							
Non-Loop-Carried							
Question:	6.05c	6.06a	6.06b	6.06a+b	6.06c	6.08a	6.08b
Table Cells Key: Pearson Correlation R Prob > R N obs.	Comp. Sci. GPA	# of High School Mathematics Courses	# of College Mathematics Courses	# of H.S. + College Mathematics courses	Mathematics GPA	Familiarity of Method Learned	Confidence in Method Learned
Variable=	QCSGPA	QHSMATH	QCOMATH	QMATHSUM	QMATHGPA	QMETHFAM	QMETHCNF
Overall	**-0.53016**	-0.01322	0.27323	0.16561	**-0.63998**	-0.13211	-0.05104
Average	**0.0053**	0.9489	0.1768	0.4188	**0.0004**	0.529	0.8044
	26	26	26	26	**26**	25	26
Annotated	**-0.51512**	0.08749	**0.39542**	0.30151	**-0.55093**	-0.0293	0.03702
Average	**0.0071**	0.6708	**0.0456**	0.1344	**0.0035**	0.8894	0.8575
	26	26	**26**	26	**26**	25	26
Manual	**-0.48998**	-0.08366	0.1542	0.04927	**-0.63988**	-0.18851	-0.10477
Average	**0.0111**	0.6845	0.452	0.8111	**0.0004**	0.3668	0.6105
	26	26	26	26	**26**	25	26
Anti-	**-0.42275**	0.08905	0.21289	0.18665	**-0.49037**	-0.15254	-0.05886
dependence	**0.0314**	0.6653	0.2964	0.3612	**0.011**	0.4667	0.7752
Average	**26**	26	26	26	**26**	25	26
Flow	**-0.5394**	-0.07629	0.23183	0.10278	**-0.66239**	-0.10283	-0.03978
dependence	**0.0045**	0.7111	0.2545	0.6173	**0.0002**	0.6248	0.847
Average	**26**	26	26	26	**26**	25	26
Output	**-0.52421**	-0.07728	**0.36021**	0.18362	**-0.64848**	-0.1053	-0.04073
dependence	**0.006**	0.7075	**0.0707**	0.3692	**0.0003**	0.6164	0.8434
Average	**26**	26	**26**	26	**26**	25	26
Anti-dep.	**-0.47077**	0.21134	**0.3676**	**0.35569**	**-0.44418**	0.02059	0.09285
Annotated	**0.0152**	0.3	**0.0647**	**0.0745**	**0.023**	0.9222	0.6519
Average	**26**	26	**26**	**26**	**26**	25	26
Anti-dep.	-0.23943	-0.05561	-0.00283	-0.03404	**-0.37406**	-0.26624	-0.18309
Manual	0.2388	0.7873	0.9891	0.8689	**0.0598**	0.1983	0.3707
Average	26	26	26	26	**26**	25	26
Flow dep.	**-0.54172**	-0.1045	0.3047	0.13263	**-0.6383**	-0.07273	-0.02816
Annotated	**0.0043**	0.6114	0.1302	0.5184	**0.0005**	0.7297	0.8914
Average	**26**	26	26	26	**26**	25	26
Flow dep.	**-0.4568**	-0.06285	0.0953	0.02399	**-0.58546**	-0.11485	-0.0444
Manual	**0.019**	0.7603	0.6433	0.9074	**0.0017**	0.5846	0.8295
Average	**26**	26	26	26	**26**	25	26
Output dep.	0.22065	0.26442	**0.58459**	**0.52408**	0.15584	-0.08808	-0.03409
Annotated	0.2787	0.1918	**0.0017**	**0.006**	0.4471	0.6754	0.8687
Average	26	26	**26**	**26**	26	25	26
Output dep.	**-0.5576**	-0.11118	0.29115	0.12017	**-0.67515**	-0.09542	-0.03692
Manual	**0.0031**	0.5887	0.149	0.5587	**0.0002**	0.65	0.8579
Average	**26**	26	26	26	**26**	25	26

(Table B.13 continued)

ERROR							
Non-Loop-Carried							
Question:	6.11a	6.11b	6.12a	6.13a	6.13b	6.14a	6.14b
Table Cells Key: Pearson Correlation R Prob > R N obs.	# of informal courses in programming	Perceived Value of Informal Learning	Perceived usefulness of "other" experiences in programming	Percentage of Time in Non-Programming Work	Percentage of Time in Seq. Programming Work	# of Programming Languages used at any time	# of Programming Languages used within last 6 months
Variable=	QNINFRML	QVINFRML	QGUSEFUL	QNPGPCT	QSPGPCT	QALLLANG	QACTLANG
Overall	0.03174	0.01913	0.24146	**0.56126**	**-0.54996**	-0.10644	0.03284
Average	0.8777	0.9675	0.2449	**0.0035**	**0.0044**	0.6048	0.8761
	26	7	25	**25**	**25**	26	25
Annotated	-0.15516	0.33113	0.11759	**0.51261**	**-0.50381**	-0.04797	0.03213
Average	0.4491	0.4682	0.5756	**0.0088**	**0.0102**	0.816	0.8788
	26	7	25	**25**	**25**	26	25
Manual	0.17514	-0.01766	0.30607	**0.54647**	**-0.53442**	-0.14371	0.02916
Average	0.3921	0.97	0.1368	**0.0047**	**0.0059**	0.4837	0.8899
	26	7	25	**25**	**25**	26	25
Anti-	0.04676	-0.09227	0.18828	**0.55155**	**-0.53713**	-0.1932	-0.06253
dependence	0.8205	0.844	0.3674	**0.0043**	**0.0056**	0.3443	0.7665
Average	26	7	25	**25**	**25**	26	25
Flow	0.03863	-0.07211	0.23664	**0.51405**	**-0.50632**	-0.07333	0.04967
dependence	0.8514	0.8779	0.2547	**0.0086**	**0.0098**	0.7218	0.8136
Average	26	7	25	**25**	**25**	26	25
Output	-0.0076	0.33113	0.27138	**0.49204**	**-0.48311**	0.02042	0.16159
dependence	0.9706	0.4682	0.1894	**0.0125**	**0.0144**	0.9211	0.4403
Average	26	7	25	**25**	**25**	26	25
Anti-dep.	-0.17303	0.33113	0.01436	**0.5026**	**-0.49143**	-0.12216	-0.05175
Annotated	0.3979	0.4682	0.9457	**0.0105**	**0.0126**	0.5522	0.8059
Average	26	7	25	**25**	**25**	26	25
Anti-dep.	0.23929	-0.11272	0.29128	**0.43176**	**-0.41849**	-0.19734	-0.05219
Manual	0.2391	0.8099	0.1577	**0.0311**	**0.0373**	0.3339	0.8043
Average	26	7	25	**25**	**25**	26	25
Flow dep.	-0.09713	0.33113	0.21475	**0.46639**	**-0.46398**	0.05634	0.12262
Annotated	0.6369	0.4682	0.3026	**0.0188**	**0.0195**	0.7846	0.5593
Average	26	7	25	**25**	**25**	26	25
Flow dep.	0.22733	-0.11272	0.24119	**0.49141**	**-0.48031**	-0.19328	-0.01426
Manual	0.2641	0.8099	0.2455	**0.0126**	**0.0151**	0.3441	0.9461
Average	26	7	25	**25**	**25**	26	25
Output dep.	-0.14564	.	0.19061	0.01388	0.00542	-0.08419	0.03462
Annotated	0.4778		0.3614	0.9475	0.9795	0.6826	0.8695
Average	26	7	25	25	25	26	25
Output dep.	0.01052	0.33113	0.2601	**0.49558**	**-0.48896**	0.03116	0.159
Manual	0.9593	0.4682	0.2092	**0.0118**	**0.0131**	0.8799	0.4478
Average	26	7	25	**25**	**25**	26	25

(Table B.13 continued)

ERROR							
Non-Loop-Carried							
Question:	6.15a	6.16a	6.17a	6.18a	6.18b	6.18d	6.18f
Table Cells Key: Pearson Correlation R Prob > R N obs.	Perceived Value of Manual Method	Perceived Value of Annotated Method	Enjoyment of working problems	Frequency of use of: IF statements	Frequency of use of: Sequential DO	Frequency of use of: CASE statement	Frequency of use of: RECURSION
Variable=	QVMANMTH	QVANNMTH	QENJOYED	QIFF	QSEQDOF	QCASEF	QRECURSE
Overall	-0.15351	-0.02582	0.00072	-0.08831	-0.07313	0.07165	0.06119
Average	0.454	0.9004	0.9972	0.6679	0.7226	0.728	0.7665
	26	26	26	26	26	26	26
Annotated	-0.12761	-0.0102	-0.04088	-0.20342	-0.08245	-0.00563	0.03765
Average	0.5344	0.9606	0.8428	0.3189	0.6889	0.9782	0.8551
	26	26	26	26	26	26	26
Manual	-0.1614	-0.0391	0.03464	0.00494	-0.05832	0.12192	0.07256
Average	0.4309	0.8496	0.8666	0.9809	0.7772	0.553	0.7246
	26	26	26	26	26	26	26
Anti-	-0.11566	0.04322	0.01234	-0.03064	-0.10832	0.00385	-0.01168
dependence	0.5737	0.8339	0.9523	0.8819	0.5984	0.9851	0.9549
Average	26	26	26	26	26	26	26
Flow	-0.2218	-0.08361	-0.00664	-0.12027	-0.08115	0.12057	0.09942
dependence	0.2762	0.6847	0.9743	0.5584	0.6935	0.5574	0.629
Average	26	26	26	26	26	26	26
Output	-0.04245	-0.0354	-0.00326	-0.09723	0.03208	0.08023	0.10057
dependence	0.8369	0.8637	0.9874	0.6365	0.8764	0.6968	0.625
Average	26	26	26	26	26	26	26
Anti-dep.	-0.12734	0.06089	0.02994	-0.20699	-0.13315	-0.11202	-0.02111
Annotated	0.5353	0.7676	0.8846	0.3103	0.5167	0.5859	0.9185
Average	26	26	26	26	26	26	26
Anti-dep.	-0.06688	0.01242	-0.00832	0.14607	-0.0495	0.11213	0.00105
Manual	0.7455	0.952	0.9678	0.4765	0.8102	0.5855	0.9959
Average	26	26	26	26	26	26	26
Flow dep.	-0.13906	-0.09012	-0.07581	-0.20286	-0.015	0.13211	0.07599
Annotated	0.4981	0.6615	0.7128	0.3203	0.942	0.52	0.7122
Average	26	26	26	26	26	26	26
Flow dep.	-0.26378	-0.08578	0.07153	-0.00357	-0.10658	0.11701	0.10476
Manual	0.1929	0.6769	0.7284	0.9862	0.6043	0.5692	0.6105
Average	26	26	26	26	26	26	26
Output dep.	0.18182	-0.02046	-0.32058	0.18731	0.00961	-0.08467	0.19509
Annotated	0.374	0.921	0.1103	0.3595	0.9628	0.6809	0.3395
Average	26	26	26	26	26	26	26
Output dep.	-0.06564	-0.03323	0.03676	-0.12172	0.03123	0.0917	0.0773
Manual	0.75	0.872	0.8585	0.5536	0.8796	0.6559	0.7074
Average	26	26	26	26	26	26	26

(Table B.13 ends)

ERROR							
Loop-Carried							
Question:	6.01a	6.02a	6.03a	6.03b	6.04aa	6.04ab	6.04a*/2
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	# of years of programming experience	Able to Juggle	Confidence in Sequential Programming	Confidence in Parallel Programming	SAT Verbal	SAT Mathematics	SAT Average
Variable=	QPROGYRS	QVJUGGLE	QCNFSQPG	QCNFPRPG	QSATV	QSATM	QSATAVG
Overall	-0.0651	**0.43511**	0.19732	-0.3009	-0.16391	-0.33312	-0.29563
Average	0.752	**0.0263**	0.3339	0.1353	0.4441	0.1117	0.1608
	26	**26**	26	26	24	24	24
Annotated	0.04498	**0.46689**	0.23725	-0.17435	-0.21734	**-0.35434**	**-0.35138**
Average	0.8273	**0.0162**	0.2432	0.3943	0.3076	**0.0893**	**0.0922**
	26	**26**	26	26	24	**24**	**24**
Manual	-0.16938	**0.35711**	0.13618	**-0.39694**	-0.0938	-0.27775	-0.20967
Average	0.4081	**0.0733**	0.5071	**0.0447**	0.6629	0.1888	0.3254
	26	**26**	26	**26**	24	24	24
Anti-	-0.04478	**0.3674**	0.30422	-0.27258	-0.20365	**-0.42294**	**-0.3715**
dependence	0.8281	**0.0648**	0.1308	0.1779	0.3399	**0.0395**	**0.0739**
Average	26	**26**	26	26	24	**24**	**24**
Flow	-0.09708	**0.41142**	0.1479	-0.21871	-0.08954	-0.26078	-0.19812
dependence	0.6371	**0.0368**	0.4709	0.2831	0.6774	0.2184	0.3534
Average	26	**26**	26	26	24	24	24
Output	0.02586	0.2499	-0.16849	-0.29102	-0.08968	0.05745	-0.05032
dependence	0.9002	0.2182	0.4106	0.1492	0.6769	0.7898	0.8154
Average	26	26	26	26	24	24	24
Anti-dep.	-0.02765	**0.43094**	0.21326	-0.20708	-0.19724	**-0.40172**	**-0.35614**
Annotated	0.8934	**0.028**	0.2955	0.3101	0.3556	**0.0517**	**0.0876**
Average	26	**26**	26	26	24	**24**	**24**
Anti-dep.	-0.05739	0.25366	**0.36285**	-0.30808	-0.18377	**-0.39**	-0.33912
Manual	0.7807	0.2112	**0.0685**	0.1257	0.39	**0.0596**	0.105
Average	26	26	**26**	26	24	**24**	24
Flow dep.	0.00359	**0.45459**	0.17696	-0.0656	-0.12974	-0.17512	-0.19284
Annotated	0.9861	**0.0196**	0.3872	0.7502	0.5457	0.4131	0.3666
Average	26	**26**	26	26	24	24	24
Flow dep.	-0.18469	**0.33233**	0.10649	**-0.34547**	-0.04206	-0.3189	-0.18436
Manual	0.3664	**0.0972**	0.6046	**0.0839**	0.8453	0.1288	0.3885
Average	26	**26**	26	**26**	24	24	24
Output dep.	0.31673	0.1299	0.21996	-0.17819	-0.26945	-0.30135	**-0.3715**
Annotated	0.1149	0.5271	0.2803	0.3838	0.2029	0.1524	**0.0739**
Average	26	26	26	26	24	24	**24**
Output dep.	-0.15137	0.19838	-0.30806	-0.21576	0.06664	0.25222	0.17448
Manual	0.4604	0.3313	0.1258	0.2898	0.757	0.2344	0.4148
Average	26	26	26	26	24	24	24

Table B.14 Selected Correlations of Loop-Carried ERROR with Questionnaire Data

ERROR							
Loop-Carried							
Question:	6.04ba	6.04bb	6.04bc	6.04b*/3	6.05a	6.05b	6.05a+b
Table Cells Key: Pearson Correlation R Prob > R N obs.	GRE Verbal	GRE Mathematics	GRE Analytical	GRE Average	# of High School Comp. Sci. courses	# of College Comp. Sci. Courses	# of H.S. + College Comp. Sci. courses
Variable=	QGREV	QGREM	QGREA	QGREAVG	QHSCS	QCOCS	QCSSUM
Overall	-0.26333	0.26897	0.21142	-0.05787	0.08947	0.10781	0.13815
Average	0.5286	0.484	0.585	0.8824	0.6638	0.6001	0.5009
	8	9	9	9	26	26	26
Annotated	-0.35915	0.29628	0.21468	-0.11594	0.10406	0.14286	0.17533
Average	0.3823	0.4389	0.5791	0.7664	0.6129	0.4863	0.3916
	8	9	9	9	26	26	26
Manual	-0.16598	0.23041	0.19739	-0.00286	0.0653	0.06105	0.08603
Average	0.6944	0.5509	0.6107	0.9942	0.7513	0.767	0.676
	8	9	9	9	26	26	26
Anti-	-0.3198	0.32579	0.26568	-0.06901	0.02457	0.29355	0.25934
dependence	0.44	0.3922	0.4896	0.86	0.9052	0.1455	0.2008
Average	8	9	9	9	26	26	26
Flow	-0.18066	0.38915	0.30516	0.07082	0.14184	-0.1272	-0.03106
dependence	0.6686	0.3006	0.4246	0.8563	0.4895	0.5358	0.8803
Average	8	9	9	9	26	26	26
Output	-0.287	-0.29214	-0.2511	-0.37623	0.04527	0.09079	0.1003
dependence	0.4907	0.4456	0.5146	0.3183	0.8262	0.6591	0.6259
Average	8	9	9	9	26	26	26
Anti-dep.	-0.52715	0.25232	0.18922	-0.23043	0.06002	0.28912	0.27454
Annotated	0.1794	0.5125	0.6258	0.5508	0.7708	0.152	0.1747
Average	8	9	9	9	26	26	26
Anti-dep.	-0.07259	0.37166	0.32071	0.10984	-0.01625	0.26142	0.21062
Manual	0.8644	0.3247	0.4001	0.7785	0.9372	0.197	0.3017
Average	8	9	9	9	26	26	26
Flow dep.	-0.17313	0.34356	0.26133	0.02124	0.20414	-0.14403	-0.01195
Annotated	0.6818	0.3653	0.497	0.9568	0.3172	0.4827	0.9538
Average	8	9	9	9	26	26	26
Flow dep.	-0.18731	0.4283	0.34389	0.11902	0.06911	-0.09941	-0.04653
Manual	0.6569	0.2501	0.3649	0.7604	0.7373	0.629	0.8214
Average	8	9	9	9	26	26	26
Output dep.	-0.29002	0.10483	0	-0.17577	-0.12694	0.29933	0.18339
Annotated	0.4859	0.7884	1	0.651	0.5366	0.1374	0.3698
Average	8	9	9	9	26	26	26
Output dep.	-0.20797	-0.37164	-0.28442	-0.35787	0.12122	-0.07083	0.00523
Manual	0.6212	0.3247	0.4582	0.3444	0.5553	0.731	0.9798
Average	8	9	9	9	26	26	26

(Table B.14 continued)

ERROR							
Loop-Carried							
Question:	6.05c	6.06a	6.06b	6.06a+b	6.06c	6.08a	6.08b
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	Comp. Sci. GPA	# of High School Mathematics Courses	# of College Mathematics Courses	# of H.S. + College Mathematics courses	Mathematics GPA	Familiarity of Method Learned	Confidence in Method Learned
Variable=	QCSGPA	QHSMATH	QCOMATH	QMATHSUM	QMATHGPA	QMETHFAM	QMETHCNF
Overall	-0.26774	-0.00996	**0.38976**	0.24141	**-0.43247**	-0.08414	-0.03082
Average	0.186	0.9615	**0.049**	0.2348	**0.0273**	0.6892	0.8812
	26	26	**26**	26	**26**	25	26
Annotated	-0.1988	0.01061	0.30506	0.19961	**-0.42304**	-0.1123	-0.07592
Average	0.3303	0.959	0.1297	0.3282	**0.0313**	0.593	0.7124
	26	26	26	26	**26**	25	26
Manual	-0.30911	-0.02967	**0.43417**	0.25814	**-0.39637**	-0.04774	0.01798
Average	0.1244	0.8856	**0.0267**	0.2029	**0.045**	0.8207	0.9305
	26	26	**26**	26	**26**	25	26
Anti-	**-0.39252**	-0.1734	**0.33243**	0.11026	**-0.59836**	-0.14429	-0.16345
dependence	**0.0473**	0.3969	**0.0971**	0.5918	**0.0012**	0.4914	0.425
Average	**26**	26	**26**	26	**26**	25	26
Flow	-0.08461	0.14603	**0.35984**	0.31289	-0.28538	-0.11275	0.00807
dependence	0.6811	0.4766	**0.071**	0.1196	0.1576	0.5916	0.9688
Average	26	26	**26**	26	26	25	26
Output	-0.12659	0.05124	0.23706	0.18006	0.07979	0.24812	0.26738
dependence	0.5377	0.8037	0.2436	0.3788	0.6984	0.2317	0.1867
Average	26	26	26	26	26	25	26
Anti-dep.	-0.25483	-0.07456	0.29801	0.14576	**-0.50873**	-0.0562	-0.06638
Annotated	0.209	0.7174	0.1392	0.4774	**0.008**	0.7896	0.7473
Average	26	26	26	26	**26**	25	26
Anti-dep.	**-0.4898**	-0.25685	0.32735	0.05864	**-0.61867**	-0.21913	-0.24625
Manual	**0.0111**	0.2053	0.1026	0.776	**0.0008**	0.2926	0.2253
Average	**26**	26	26	26	**26**	25	26
Flow dep.	-0.02423	0.18324	**0.33175**	0.31666	-0.19509	-0.09046	0.02808
Annotated	0.9065	0.3703	**0.0978**	0.115	0.3395	0.6672	0.8917
Average	26	26	**26**	26	26	25	26
Flow dep.	-0.13473	0.09698	**0.35378**	0.2806	**-0.34579**	-0.12369	-0.01184
Manual	0.5117	0.6374	**0.0762**	0.165	**0.0836**	0.5558	0.9542
Average	26	26	**26**	26	**26**	25	26
Output dep.	-0.27696	-0.18977	-0.05142	-0.14266	-0.32277	-0.21473	-0.29098
Annotated	0.1708	0.3531	0.803	0.4869	0.1078	0.3027	0.1493
Average	26	26	26	26	26	25	26
Output dep.	0.01919	0.16333	0.28717	0.27684	0.26978	**0.41514**	**0.45595**
Manual	0.9259	0.4253	0.1549	0.171	0.1826	**0.0391**	**0.0192**
Average	26	26	26	26	26	**25**	**26**

(Table B.14 continued)

ERROR							
Loop-Carried							
Question:	6.11a	6.11b	6.12a	6.13a	6.13b	6.14a	6.14b
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	# of informal courses in programming	Perceived Value of Informal Learning	Perceived usefulness of "other" experiences in programming	Percentage of Time in Non-Programming Work	Percentage of Time in Seq. Programming Work	# of Programming Languages used at any time	# of Programming Languages used within last 6 months
Variable=	QNINFRML	QVINFRML	QGUSEFUL	QNPGPCT	QSPGPCT	QALLLANG	QACTLANG
Overall	0.06262	-0.1653	0.24499	**0.48189**	**-0.43104**	-0.17122	0.01735
Average	0.7612	0.7232	0.2379	**0.0147**	**0.0315**	0.403	0.9344
	26	7	25	**25**	**25**	26	25
Annotated	0.06725	-0.35602	0.2151	**0.38653**	**-0.34025**	-0.14323	0.00944
Average	0.7441	0.4332	0.3018	**0.0563**	**0.0961**	0.4852	0.9643
	26	7	25	**25**	**25**	26	25
Manual	0.05135	0.08671	0.2493	**0.53039**	**-0.48006**	-0.18142	0.02358
Average	0.8033	0.8533	0.2295	**0.0064**	**0.0152**	0.3751	0.9109
	26	7	25	**25**	**25**	26	25
Anti-	0.145	0.04347	0.28011	**0.39693**	**-0.34979**	-0.07981	0.12816
dependence	0.4797	0.9263	0.175	**0.0495**	**0.0865**	0.6984	0.5415
Average	26	7	25	**25**	**25**	26	25
Flow	0.08787	-0.35306	0.26876	**0.48918**	**-0.45376**	-0.17388	-0.0056
dependence	0.6695	0.4373	0.1939	**0.0131**	**0.0227**	0.3956	0.9788
Average	26	7	25	**25**	**25**	26	25
Output	-0.29319	-0.13245	-0.15794	0.21735	-0.16694	-0.24699	-0.2446
dependence	0.1461	0.7771	0.4508	0.2967	0.4251	0.2238	0.2386
Average	26	7	25	25	25	26	25
Anti-dep.	0.12483	-0.1776	0.24009	**0.37544**	-0.32544	-0.10244	0.08101
Annotated	0.5435	0.7032	0.2477	**0.0644**	0.1124	0.6185	0.7003
Average	26	7	25	**25**	25	26	25
Anti-dep.	0.14826	0.26512	0.28969	**0.3689**	-0.33083	-0.0457	0.16169
Manual	0.4698	0.5656	0.1601	**0.0696**	0.1062	0.8246	0.44
Average	26	7	25	**25**	25	26	25
Flow dep.	0.06589	-0.50634	0.22857	**0.44283**	**-0.41618**	-0.1365	0.00326
Annotated	0.7491	0.2462	0.2718	**0.0266**	**0.0385**	0.5061	0.9877
Average	26	7	25	**25**	**25**	26	25
Flow dep.	0.10089	-0.12083	0.2825	**0.49723**	**-0.45602**	-0.19378	-0.01361
Manual	0.6238	0.7964	0.1712	**0.0114**	**0.022**	0.3429	0.9485
Average	26	7	25	**25**	**25**	26	25
Output dep.	-0.16575	-0.13245	-0.10817	-0.09429	0.13498	-0.1343	-0.19654
Annotated	0.4184	0.7771	0.6068	0.6539	0.52	0.5131	0.3464
Average	26	7	25	25	25	26	25
Output dep.	-0.22518	-0.13245	-0.11089	0.29069	-0.25894	-0.19272	-0.15917
Manual	0.2687	0.7771	0.5977	0.1586	0.2113	0.3456	0.4473
Average	26	7	25	25	25	26	25

(Table B.14 continued)

ERROR							
Loop-Carried							
Question:	6.15a	6.16a	6.17a	6.18a	6.18b	6.18d	6.18f
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	Perceived Value of Manual Method	Perceived Value of Annotated Method	Enjoyment of working problems	Frequency of use of: IF statements	Frequency of use of: Sequential DO	Frequency of use of: CASE statement	Frequency of use of: RECURSION
Variable=	QVMANMTH	QVANNMTH	QENJOYED	QIFF	QSEQDOF	QCASEF	QRECURSE
Overall	0.0649	**0.41681**	0.06598	0.28037	-0.25285	0.09826	0.29273
Average	0.7528	**0.0341**	0.7488	0.1653	0.2127	0.633	0.1467
	26	**26**	26	26	26	26	26
Annotated	-0.02467	**0.35673**	-0.04674	0.28249	-0.08607	0.15799	0.18745
Average	0.9048	**0.0736**	0.8206	0.162	0.6759	0.4408	0.3592
	26	**26**	26	26	26	26	26
Manual	0.1485	**0.43351**	0.17284	0.24865	**-0.39459**	0.02757	**0.36816**
Average	0.4691	**0.0269**	0.3985	0.2206	**0.0461**	0.8936	**0.0642**
	26	**26**	26	26	**26**	26	**26**
Anti-	-0.01381	**0.34295**	0.03913	0.25499	-0.03445	0.24723	0.2051
dependence	0.9466	**0.0863**	0.8495	0.2087	0.8673	0.2234	0.3148
Average	26	**26**	26	26	26	26	26
Flow	0.03024	**0.33999**	0.03196	0.29341	**-0.37644**	-0.00788	**0.3504**
dependence	0.8834	**0.0892**	0.8768	0.1457	**0.058**	0.9695	**0.0793**
Average	26	**26**	26	26	**26**	26	**26**
Output	0.30766	**0.40378**	0.1622	0.03754	-0.28775	-0.13928	0.09517
dependence	0.1263	**0.0408**	0.4286	0.8555	0.154	0.4974	0.6438
Average	26	**26**	26	26	26	26	26
Anti-dep.	-0.05206	0.2838	-0.02754	0.25316	0.02229	0.21336	0.14429
Annotated	0.8006	0.16	0.8938	0.2121	0.9139	0.2953	0.4819
Average	26	26	26	26	26	26	26
Anti-dep.	0.02863	**0.36286**	0.10518	0.22493	-0.09053	0.25226	0.24408
Manual	0.8896	**0.0685**	0.6091	0.2693	0.6601	0.2138	0.2295
Average	26	**26**	26	26	26	26	26
Flow dep.	0.0579	0.32347	-0.02833	0.27891	-0.2261	0.03347	0.25301
Annotated	0.7787	0.107	0.8907	0.1676	0.2667	0.871	0.2124
Average	26	26	26	26	26	26	26
Flow dep.	0.00096	0.32466	0.08684	0.28042	**-0.48607**	-0.0468	**0.41165**
Manual	0.9963	0.1056	0.6732	0.1653	**0.0118**	0.8204	**0.0367**
Average	26	26	26	26	**26**	26	**26**
Output dep.	-0.12932	0.29271	-0.10226	0.09091	0.04022	0.15233	-0.043
Annotated	0.5289	0.1467	0.6191	0.6587	0.8453	0.4575	0.8348
Average	26	26	26	26	26	26	26
Output dep.	**0.40816**	0.27359	0.2345	-0.01066	**-0.33599**	-0.23793	0.12796
Manual	**0.0385**	0.1762	0.2489	0.9588	**0.0933**	0.2418	0.5333
Average	**26**	26	26	26	**26**	26	26

(Table B.14 ends)

TASKTIME							
Non-Loop-Carried							
Question:	6.01a	6.02a	6.03a	6.03b	6.04aa	6.04ab	6.04a*/2
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	# of years of programming experience	Able to Juggle	Confidence in Sequential Programming	Confidence in Parallel Programming	SAT Verbal	SAT Mathematics	SAT Average
Variable=	QPROGYRS	QVJUGGLE	QCNFSQPG	QCNFPRPG	QSATV	QSATM	QSATAVG
Overall	-0.29973	0.11692	-0.07285	0.05713	0.04192	0.02252	0.04648
Average	0.1368	0.5695	0.7236	0.7816	0.8458	0.9168	0.8293
	26	26	26	26	24	24	24
Annotated	**-0.45803**	**0.34786**	-0.21213	0.01659	-0.1796	-0.04863	-0.17686
Average	**0.0186**	**0.0816**	0.2982	0.9359	0.401	0.8215	0.4084
	26	**26**	26	26	24	24	24
Manual	-0.16209	-0.01147	0.0122	0.05489	0.12691	0.06704	0.14016
Average	0.4289	0.9556	0.9528	0.79	0.5546	0.7556	0.5136
	26	26	26	26	24	24	24
Anti-	**-0.34843**	0.01318	-0.22937	-0.05459	-0.14262	0.17623	-0.04058
dependence	**0.0811**	0.9491	0.2597	0.7911	0.5062	0.4101	0.8507
Average	**26**	26	26	26	24	24	24
Flow	-0.08376	-0.01296	0.12961	0.11886	0.20863	-0.11528	0.13614
dependence	0.6906	0.951	0.5369	0.5715	0.3394	0.6004	0.5357
Average	25	25	25	25	23	23	23
Output	-0.22499	0.25483	-0.13807	0.06181	0.06139	-0.02972	0.03891
dependence	0.2691	0.209	0.5012	0.7642	0.7757	0.8904	0.8567
Average	26	26	26	26	24	24	24
Anti-dep.	**-0.43919**	0.06953	**-0.62637**	-0.24135	-0.07728	0.31498	0.04708
Annotated	**0.0281**	0.7412	**0.0008**	0.2451	0.726	0.1432	0.8311
Average	**25**	25	**25**	25	23	23	23
Anti-dep.	-0.15614	-0.09458	0.15333	0.08059	-0.09414	0.02989	-0.06703
Manual	0.4561	0.6529	0.4643	0.7018	0.6692	0.8923	0.7612
Average	25	25	25	25	23	23	23
Flow dep.	-0.18645	0.13127	0.28176	0.14246	-0.16787	-0.22049	-0.22171
Annotated	0.3722	0.5317	0.1724	0.4969	0.4439	0.312	0.3093
Average	25	25	25	25	23	23	23
Flow dep.	-0.01238	-0.02219	0.0684	0.08182	0.28719	-0.06607	0.22089
Manual	0.9542	0.918	0.7508	0.7039	0.195	0.7702	0.3232
Average	24	24	24	24	22	22	22
Output dep.	-0.19723	**0.37063**	0.11541	0.19122	-0.06995	-0.13978	-0.12505
Annotated	0.3342	**0.0623**	0.5745	0.3494	0.7453	0.5148	0.5604
Average	26	**26**	26	26	24	24	24
Output dep.	-0.14228	0.0585	-0.23691	-0.05125	0.14904	0.03267	0.13844
Manual	0.4975	0.7812	0.2542	0.8078	0.4973	0.8824	0.5287
Average	25	25	25	25	23	23	23

Table B.15 Selected Correlations of Non-Loop-Carried TASKTIME with Questionnaire Data

TASKTIME							
Non-Loop-Carried							
Question:	6.04ba	6.04bb	6.04bc	6.04b*/3	6.05a	6.05b	6.05a+b
Table Cells Key: Pearson Correlation R Prob > R N obs.	GRE Verbal	GRE Mathematics	GRE Analytical	GRE Average	# of High School Comp. Sci. courses	# of College Comp. Sci. Courses	# of H.S. + College Comp. Sci. courses
Variable=	QGREV	QGREM	QGREA	QGREAVG	QHSCS	QCOCS	QCSSUM
Overall	-0.16163	**-0.86831**	**-0.8618**	**-0.63993**	-0.29666	-0.18784	-0.31577
Average	0.7022	**0.0024**	**0.0028**	**0.0634**	0.1411	0.3581	0.1161
	8	**9**	**9**	**9**	26	26	26
Annotated	-0.34689	**-0.74515**	**-0.8124**	**-0.73535**	-0.28532	-0.18725	-0.30923
Average	0.3999	**0.0212**	**0.0078**	**0.024**	0.1577	0.3597	0.1242
	8	**9**	**9**	**9**	26	26	26
Manual	-0.05228	**-0.84776**	**-0.80188**	-0.52383	-0.23018	-0.15196	-0.25022
Average	0.9021	**0.0039**	**0.0093**	0.1478	0.258	0.4586	0.2176
	8	**9**	**9**	9	26	26	26
Anti-	-0.01891	**-0.72829**	**-0.67327**	-0.36038	-0.26592	-0.26995	**-0.36826**
dependence	0.9646	**0.0261**	**0.0468**	0.3407	0.1892	0.1823	**0.0642**
Average	8	**9**	**9**	9	26	26	**26**
Flow	-0.31188	**-0.71144**	**-0.6894**	**-0.67502**	-0.16293	0.04072	-0.05144
dependence	0.452	**0.0316**	**0.0399**	**0.046**	0.4365	0.8468	0.8071
Average	8	**9**	**9**	**9**	25	25	25
Output	-0.14717	**-0.89856**	**-0.95593**	**-0.70566**	-0.21718	-0.15111	-0.24258
dependence	0.728	**0.001**	**0.0001**	**0.0337**	0.2866	0.4612	0.2325
Average	8	**9**	**9**	**9**	26	26	26
Anti-dep.	-0.24987	**-0.81085**	**-0.71039**	**-0.59234**	-0.24489	-0.1839	-0.284
Annotated	0.5506	**0.008**	**0.032**	**0.0928**	0.2381	0.3789	0.1689
Average	8	**9**	**9**	**9**	25	25	25
Anti-dep.	0.30298	-0.36454	-0.38426	0.05001	-0.21512	-0.25514	-0.32908
Manual	0.4657	0.3348	0.3072	0.8983	0.3018	0.2184	0.1082
Average	8	9	9	9	25	25	25
Flow dep.	-0.50326	-0.12077	-0.21019	-0.46587	-0.05923	0.01134	-0.02162
Annotated	0.2036	0.7569	0.5873	0.2063	0.7785	0.9571	0.9183
Average	8	9	9	9	25	25	25
Flow dep.	-0.01073	**-0.8697**	**-0.76541**	-0.54476	-0.16707	0.03082	-0.06248
Manual	0.9799	**0.0023**	**0.0162**	0.1294	0.4352	0.8863	0.7718
Average	8	**9**	**9**	9	24	24	24
Output dep.	0.19807	-0.31222	-0.57468	-0.2578	-0.17927	-0.11879	-0.19525
Annotated	0.6382	0.4134	0.1055	0.503	0.3809	0.5633	0.3391
Average	8	9	9	9	26	26	26
Output dep.	-0.25812	**-0.93315**	**-0.88868**	**-0.72741**	-0.17062	-0.11	-0.18259
Manual	0.5371	**0.0002**	**0.0014**	**0.0263**	0.4148	0.6007	0.3823
Average	8	**9**	**9**	**9**	25	25	25

(Table B.15 continued)

TASKTIME							
Non-Loop-Carried							
Question:	6.05c	6.06a	6.06b	6.06a+b	6.06c	6.08a	6.08b
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	Comp. Sci. GPA	# of High School Mathematics Courses	# of College Mathematics Courses	# of H.S. + College Mathematics courses	Mathematics GPA	Familiarity of Method Learned	Confidence in Method Learned
Variable=	QCSGPA	QHSMATH	QCOMATH	QMATHSUM	QMATHGPA	QMETHFAM	QMETHCNF
Overall	0.15914	**-0.33213**	-0.28189	**-0.37139**	0.04533	-0.08367	-0.04905
Average	0.4374	**0.0974**	0.163	**0.0618**	0.826	0.6909	0.8119
	26	**26**	26	**26**	26	25	26
Annotated	0.02651	-0.15695	0.08544	-0.03683	-0.10603	-0.24428	-0.16879
Average	0.8977	0.4439	0.6781	0.8582	0.6062	0.2393	0.4098
	26	26	26	26	26	25	26
Manual	0.18857	**-0.34256**	**-0.40944**	**-0.45832**	0.11038	0.0225	0.02413
Average	0.3562	**0.0867**	**0.0378**	**0.0185**	0.5914	0.915	0.9069
	26	**26**	**26**	**26**	26	25	26
Anti-	0.07838	-0.09816	-0.07498	-0.10448	0.0441	-0.027	0.01276
dependence	0.7035	0.6333	0.7158	0.6115	0.8306	0.8981	0.9507
Average	26	26	26	26	26	25	26
Flow	0.2582	-0.33492	-0.26648	**-0.35826**	0.05792	-0.11962	-0.15879
dependence	0.2127	0.1017	0.1979	**0.0787**	0.7833	0.5777	0.4484
Average	25	25	25	**25**	25	24	25
Output	0.0145	-0.24349	-0.24183	-0.29457	0.03734	0.0632	0.11683
dependence	0.9439	0.2307	0.234	0.1441	0.8563	0.7641	0.5698
Average	26	26	26	26	26	25	26
Anti-dep.	-0.20294	0.18674	0.16014	0.20686	0.02425	0.20722	0.23124
Annotated	0.3306	0.3714	0.4445	0.3211	0.9084	0.3313	0.2661
Average	25	25	25	25	25	24	25
Anti-dep.	0.16835	-0.22699	-0.18635	-0.25074	-0.00639	-0.16413	-0.12881
Manual	0.4211	0.2752	0.3725	0.2267	0.9758	0.4435	0.5395
Average	25	25	25	25	25	24	25
Flow dep.	0.30432	-0.16985	0.23467	0.04575	0.0397	-0.22225	-0.16355
Annotated	0.1391	0.417	0.2588	0.8281	0.8506	0.2966	0.4347
Average	25	25	25	25	25	24	25
Flow dep.	0.13395	**-0.34515**	**-0.40512**	**-0.4508**	-0.02334	-0.06468	-0.12441
Manual	0.5326	**0.0986**	**0.0495**	**0.027**	0.9138	0.7694	0.5624
Average	24	**24**	**24**	**24**	24	23	24
Output dep.	0.08315	-0.22117	-0.14719	-0.2216	-0.14762	**-0.4477**	**-0.44168**
Annotated	0.6863	0.2776	0.4731	0.2766	0.4717	**0.0248**	**0.0239**
Average	26	26	26	26	26	**25**	**26**
Output dep.	-0.09912	-0.16617	-0.18645	-0.2111	0.11333	**0.45051**	**0.43193**
Manual	0.6373	0.4273	0.3722	0.3111	0.5896	**0.0272**	**0.0311**
Average	25	25	25	25	25	**24**	**25**

(Table B.15 continued)

TASKTIME							
Non-Loop-Carried							
Question:	6.11a	6.11b	6.12a	6.13a	6.13b	6.14a	6.14b
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	# of informal courses in programming	Perceived Value of Informal Learning	Perceived usefulness of "other" experiences in programming	Percentage of Time in Non-Programming Work	Percentage of Time in Seq. Programming Work	# of Programming Languages used at any time	# of Programming Languages used within last 6 months
Variable=	QNINFRML	QVINFRML	QGUSEFUL	QNPGPCT	QSPGPCT	QALLLANG	QACTLANG
Overall	0.03613	0.05093	0.02209	-0.09544	0.04285	-0.16218	-0.07958
Average	0.8609	0.9136	0.9165	0.65	0.8388	0.4286	0.7053
	26	7	25	25	25	26	25
Annotated	0.08641	0.23245	0.01836	0.19194	-0.2618	-0.2025	-0.00138
Average	0.6747	0.616	0.9306	0.358	0.2062	0.3212	0.9948
	26	7	25	25	25	26	25
Manual	0.03554	-0.08666	0.02315	-0.20706	0.1742	-0.11658	-0.0968
Average	0.8632	0.8534	0.9125	0.3206	0.405	0.5706	0.6453
	26	7	25	25	25	26	25
Anti-	0.15557	0.17107	-0.11133	0.09002	-0.10901	0.06287	0.21718
dependence	0.4479	0.7138	0.5962	0.6687	0.604	0.7603	0.297
Average	26	7	25	25	25	26	25
Flow	-0.14279	-0.28292	0.12501	**-0.36237**	**0.34618**	-0.03238	-0.09182
dependence	0.4959	0.5387	0.5606	**0.0818**	**0.0975**	0.8779	0.6696
Average	25	7	24	**24**	**24**	25	24
Output	-0.03309	0.08843	-0.02514	0.04861	-0.14909	**-0.46529**	**-0.42535**
dependence	0.8725	0.8505	0.9051	0.8175	0.4769	**0.0166**	**0.034**
Average	26	7	25	25	25	**26**	**25**
Anti-dep.	-0.0047	-0.07573	-0.3314	0.27043	-0.2453	-0.1599	-0.03134
Annotated	0.9822	0.8718	0.1137	0.2012	0.248	0.4452	0.8844
Average	25	7	24	24	24	25	24
Anti-dep.	0.28071	0.22495	0.1247	0.03712	-0.08842	0.19804	0.32881
Manual	0.1741	0.6683	0.5615	0.8633	0.6812	0.3426	0.1167
Average	25	6	24	24	24	25	24
Flow dep.	-0.08539	0.2109	0.05156	0.11573	-0.18582	0.08446	0.17144
Annotated	0.6849	0.6499	0.8109	0.5902	0.3847	0.6881	0.4231
Average	25	7	24	24	24	25	24
Flow dep.	-0.04601	-0.60376	0.16545	**-0.40246**	**0.40824**	-0.09681	-0.15104
Manual	0.8309	0.2044	0.4506	**0.0569**	**0.0531**	0.6527	0.4915
Average	24	6	23	**23**	**23**	24	23
Output dep.	0.18479	0.38627	**0.39943**	-0.02296	-0.11535	-0.27845	-0.17699
Annotated	0.3661	0.392	**0.0479**	0.9132	0.583	0.1684	0.3974
Average	26	7	**25**	25	25	26	25
Output dep.	-0.17078	-0.27029	-0.27758	0.12676	-0.17317	**-0.38941**	**-0.39026**
Manual	0.4144	0.5577	0.1891	0.555	0.4184	**0.0543**	**0.0594**
Average	25	7	24	24	24	**25**	**24**

(Table B.15 continued)

TASKTIME							
Non-Loop-Carried							
Question:	6.15a	6.16a	6.17a	6.18a	6.18b	6.18d	6.18f
Table Cells Key: Pearson Correlation R Prob > R N obs.	Perceived Value of Manual Method	Perceived Value of Annotated Method	Enjoyment of working problems	Frequency of use of: IF statements	Frequency of use of: Sequential DO	Frequency of use of: CASE statement	Frequency of use of: RECURSION
Variable=	QVMANMTH	QVANNMTH	QENJOYED	QIFF	QSEQDOF	QCASEF	QRECURSE
Overall	0.27583	0.193	0.18974	0.10905	0.15273	-0.21876	-0.08341
Average	0.1726	0.3448	0.3532	0.5959	0.4564	0.283	0.6854
	26	26	26	26	26	26	26
Annotated	0.32007	0.21478	0.30545	0.12561	0.03317	-0.14608	-0.07683
Average	0.1109	0.292	0.1292	0.5409	0.8722	0.4764	0.7091
	26	26	26	26	26	26	26
Manual	0.18785	0.13767	0.10793	0.08987	0.17578	-0.20346	-0.06473
Average	0.3581	0.5024	0.5997	0.6624	0.3904	0.3188	0.7534
	26	26	26	26	26	26	26
Anti-	-0.01086	-0.07667	0.26167	-0.17362	0.17575	**-0.37983**	-0.21361
dependence	0.958	0.7097	0.1966	0.3963	0.3904	**0.0556**	0.2947
Average	26	26	26	26	26	**26**	26
Flow	0.19401	0.22965	-0.11807	0.15931	0.05909	-0.14359	0.01968
dependence	0.3527	0.2695	0.574	0.4469	0.779	0.4935	0.9256
Average	25	25	25	25	25	25	25
Output	**0.50685**	0.31492	0.27693	0.25214	0.08451	0.11371	0.02691
dependence	**0.0082**	0.1171	0.1708	0.214	0.6815	0.5802	0.8962
Average	**26**	26	26	26	26	26	26
Anti-dep.	0.1657	0.10722	0.28044	-0.26202	0.09887	-0.29863	-0.21718
Annotated	0.4286	0.61	0.1745	0.2058	0.6382	0.147	0.297
Average	25	25	25	25	25	25	25
Anti-dep.	-0.19214	-0.19896	0.14751	-0.0618	0.15562	-0.31128	-0.14105
Manual	0.3575	0.3404	0.4816	0.7692	0.4576	0.1299	0.5013
Average	25	25	25	25	25	25	25
Flow dep.	0.05365	0.16877	0.20865	0.15321	-0.20016	-0.24273	0.02344
Annotated	0.799	0.42	0.3169	0.4647	0.3374	0.2424	0.9115
Average	25	25	25	25	25	25	25
Flow dep.	0.17462	0.16958	-0.18475	0.14555	0.14836	-0.03979	0.02205
Manual	0.4144	0.4283	0.3875	0.4974	0.489	0.8536	0.9186
Average	24	24	24	24	24	24	24
Output dep.	**0.38665**	0.22717	0.00168	**0.49009**	0.03726	0.2748	0.08873
Annotated	**0.051**	0.2644	0.9935	**0.011**	0.8566	0.1743	0.6664
Average	**26**	26	26	**26**	26	26	26
Output dep.	**0.36504**	0.23325	**0.3367**	-0.02829	0.08069	-0.02832	-0.02021
Manual	**0.0728**	0.2618	**0.0998**	0.8932	0.7014	0.8931	0.9236
Average	**25**	25	**25**	25	25	25	25

(Table B.15 ends)

TASKTIME							
Loop-Carried							
Question:	6.01a	6.02a	6.03a	6.03b	6.04aa	6.04ab	6.04a*/2
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	# of years of programming experience	Able to Juggle	Confidence in Sequential Programming	Confidence in Parallel Programming	SAT Verbal	SAT Mathematics	SAT Average
Variable=	QPROGYRS	QVJUGGLE	QCNFSQPG	QCNFPRPG	QSATV	QSATM	QSATAVG
Overall	-0.01918	**0.44504**	0.08975	-0.00648	0.02274	-0.19011	-0.06884
Average	0.9259	**0.0227**	0.6628	0.9749	0.916	0.3736	0.7493
	26	**26**	26	26	24	24	24
Annotated	-0.01066	**0.4966**	0.10977	-0.10347	0.05936	-0.18731	-0.03609
Average	0.9588	**0.0099**	0.5935	0.615	0.7829	0.3808	0.867
	26	**26**	26	26	24	24	24
Manual	-0.03199	0.24234	0.00411	0.08978	-0.00512	-0.20778	-0.10098
Average	0.8767	0.2329	0.9841	0.6627	0.981	0.3299	0.6387
	26	26	26	26	24	24	24
Anti-	-0.14307	0.29565	-0.00085	-0.19553	0.09244	-0.27597	-0.04889
dependence	0.4856	0.1425	0.9967	0.3384	0.6675	0.1918	0.8205
Average	26	26	26	26	24	24	24
Flow	0.14166	**0.48278**	0.12246	0.29195	-0.03623	0.10373	0.0171
dependence	0.49	**0.0125**	0.5512	0.1478	0.8665	0.6296	0.9368
Average	26	**26**	26	26	24	24	24
Output	-0.03396	0.30769	0.13639	-0.11	-0.00628	-0.28773	-0.13914
dependence	0.8692	0.1262	0.5065	0.5927	0.9768	0.1728	0.5167
Average	26	26	26	26	24	24	24
Anti-dep.	-0.12177	0.33047	0.01479	-0.18059	0.26968	-0.26004	0.10236
Annotated	0.562	0.1066	0.9441	0.3877	0.2133	0.2308	0.6421
Average	25	25	25	25	23	23	23
Anti-dep.	-0.14758	0.18896	-0.02507	-0.16719	-0.10738	-0.19115	-0.17928
Manual	0.4814	0.3657	0.9053	0.4244	0.6258	0.3823	0.4131
Average	25	25	25	25	23	23	23
Flow dep.	0.08287	**0.41877**	0.01192	0.03498	0.04717	0.06206	0.06936
Annotated	0.6874	**0.0332**	0.9539	0.8653	0.8268	0.7733	0.7474
Average	26	**26**	26	26	24	24	24
Flow dep.	0.14996	**0.37013**	0.22898	**0.43552**	-0.10199	0.06625	-0.05742
Manual	0.4743	**0.0686**	0.2709	**0.0296**	0.6433	0.7639	0.7947
Average	25	**25**	25	**25**	23	23	23
Output dep.	0.0179	0.26253	0.23268	-0.07136	-0.18337	-0.22916	-0.26401
Annotated	0.9308	0.1951	0.2527	0.729	0.3911	0.2814	0.2125
Average	26	26	26	26	24	24	24
Output dep.	-0.13275	0.24234	-0.0885	-0.1597	0.18459	-0.26109	0.03718
Manual	0.518	0.2329	0.6673	0.4358	0.3879	0.2178	0.8631
Average	26	26	26	26	24	24	24

Table B.16 Selected Correlations of Loop-Carried TASKTIME with Questionnaire Data

TASKTIME							
Loop-Carried							
Question:	6.04ba	6.04bb	6.04bc	6.04b*/3	6.05a	6.05b	6.05a+b
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	GRE Verbal	GRE Mathematics	GRE Analytical	GRE Average	# of High School Comp. Sci. courses	# of College Comp. Sci. Courses	# of H.S. + College Comp. Sci. courses
Variable=	QGREV	QGREM	QGREA	QGREAVG	QHSCS	QCOCS	QCSSUM
Overall	-0.04247	-0.39431	-0.24758	-0.07108	**-0.34192**	0.30953	0.0773
Average	0.9205	0.2936	0.5207	0.8558	**0.0873**	0.1239	0.7074
	8	9	9	9	**26**	26	26
Annotated	-0.20632	-0.14287	-0.01048	0.06419	-0.2718	0.32929	0.13127
Average	0.624	0.7138	0.9787	0.8697	0.1792	0.1005	0.5227
	8	9	9	9	26	26	26
Manual	0.03239	-0.5569	-0.44238	-0.2094	**-0.4289**	0.17187	-0.08455
Average	0.9393	0.1193	0.2331	0.5887	**0.0288**	0.4012	0.6813
	8	9	9	9	**26**	26	26
Anti-	0.22406	-0.37111	-0.0991	-0.00292	-0.29685	0.09037	-0.0825
dependence	0.5937	0.3255	0.7998	0.994	0.1409	0.6607	0.6887
Average	8	9	9	9	26	26	26
Flow	-0.0497	-0.0676	-0.03968	0.17997	-0.28574	0.309	0.10682
dependence	0.907	0.8628	0.9193	0.6431	0.1571	0.1245	0.6035
Average	8	9	9	9	26	26	26
Output	-0.45531	**-0.62415**	**-0.58233**	-0.56087	-0.18305	**0.37451**	0.21653
dependence	0.2569	**0.0724**	**0.0999**	0.1162	0.3708	**0.0594**	0.288
Average	8	**9**	**9**	9	26	**26**	26
Anti-dep.	0.02403	-0.28073	-0.01748	-0.03904	-0.23654	0.02615	-0.12099
Annotated	0.955	0.4643	0.9644	0.9206	0.2549	0.9013	0.5646
Average	8	9	9	9	25	25	25
Anti-dep.	0.4743	-0.35649	-0.07685	0.10695	-0.24153	0.08585	-0.05481
Manual	0.2822	0.3861	0.8565	0.801	0.2448	0.6833	0.7947
Average	7	8	8	8	25	25	25
Flow dep.	-0.53512	0.20191	0.18537	0.31212	-0.15725	0.18996	0.07548
Annotated	0.1717	0.6024	0.633	0.4135	0.443	0.3526	0.714
Average	8	9	9	9	26	26	26
Flow dep.	0.16246	-0.38718	-0.28936	-0.06682	-0.28726	**0.34977**	0.15245
Manual	0.7278	0.3433	0.487	0.8751	0.1638	**0.0865**	0.4669
Average	7	8	8	8	25	**25**	25
Output dep.	-0.30143	**-0.60109**	-0.53147	**-0.61074**	-0.1489	**0.4481**	0.29648
Annotated	0.4681	**0.0869**	0.1409	**0.0806**	0.4678	**0.0217**	0.1414
Average	8	**9**	9	**9**	26	**26**	26
Output dep.	-0.44708	-0.33104	-0.33579	-0.2359	-0.22998	0.17897	0.02748
Manual	0.2667	0.3842	0.377	0.5412	0.2584	0.3817	0.894
Average	8	9	9	9	26	26	26

(Table B.16 continued)

TASKTIME							
Loop-Carried							
Question:	6.05c	6.06a	6.06b	6.06a+b	6.06c	6.08a	6.08b
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	Comp. Sci. GPA	# of High School Mathematics Courses	# of College Mathematics Courses	# of H.S. + College Mathematics courses	Mathematics GPA	Familiarity of Method Learned	Confidence in Method Learned
Variable=	QCSGPA	QHSMATH	QCOMATH	QMATHSUM	QMATHGPA	QMETHFAM	QMETHCNF
Overall	0.10174	**-0.49556**	**-0.36704**	**-0.52016**	-0.10553	-0.16585	-0.27647
Average	0.6209	**0.01**	**0.0651**	**0.0065**	0.6079	0.4282	0.1715
	26	**26**	**26**	**26**	26	25	26
Annotated	0.09198	**-0.46422**	-0.23235	**-0.41657**	-0.12234	-0.13711	-0.21244
Average	0.655	**0.0169**	0.2534	**0.0343**	0.5516	0.5134	0.2975
	26	**26**	26	**26**	26	25	26
Manual	0.04408	**-0.44135**	**-0.56608**	**-0.61495**	-0.09715	-0.13041	-0.26562
Average	0.8307	**0.024**	**0.0026**	**0.0008**	0.6368	0.5344	0.1897
	26	**26**	**26**	**26**	26	25	26
Anti-	-0.12241	**-0.56581**	-0.21854	**-0.46672**	-0.23359	-0.2017	-0.19742
dependence	0.5514	**0.0026**	0.2835	**0.0162**	0.2508	0.3336	0.3337
Average	26	**26**	26	**26**	26	25	26
Flow	0.28387	-0.27743	-0.32927	**-0.3697**	0.03405	-0.15074	-0.3215
dependence	0.1599	0.17	0.1005	**0.063**	0.8688	0.472	0.1092
Average	26	26	26	**26**	26	25	26
Output	0.12706	-0.30772	-0.29796	**-0.36742**	-0.02873	-0.02486	-0.13401
dependence	0.5362	0.1262	0.1393	**0.0648**	0.8892	0.9061	0.514
Average	26	26	26	**26**	26	25	26
Anti-dep.	-0.19978	**-0.53183**	-0.17229	**-0.42252**	**-0.33763**	-0.08978	-0.06373
Annotated	0.3383	**0.0062**	0.4102	**0.0354**	**0.0988**	0.6765	0.7622
Average	25	**25**	25	**25**	**25**	24	25
Anti-dep.	0.06987	**-0.44172**	-0.12916	**-0.35402**	0.01668	-0.30342	-0.28239
Manual	0.74	**0.0271**	0.5383	**0.0825**	0.9369	0.1495	0.1714
Average	25	**25**	25	**25**	25	24	25
Flow dep.	**0.33183**	-0.04652	-0.00599	-0.03077	0.15877	-0.09466	-0.13724
Annotated	**0.0977**	0.8215	0.9769	0.8814	0.4385	0.6526	0.5038
Average	**26**	26	26	26	26	25	26
Flow dep.	0.08291	**-0.41611**	**-0.50494**	**-0.54932**	-0.1606	-0.17587	**-0.40132**
Manual	0.6936	**0.0386**	**0.01**	**0.0045**	0.4431	0.4111	**0.0468**
Average	25	**25**	**25**	**25**	25	24	**25**
Output dep.	0.12022	**-0.40939**	**-0.33398**	**-0.44922**	-0.02508	-0.09435	-0.22102
Annotated	0.5586	**0.0378**	**0.0954**	**0.0213**	0.9032	0.6537	0.2779
Average	26	**26**	**26**	**26**	26	25	26
Output dep.	0.07011	-0.06892	-0.19765	-0.16532	-0.02727	0.07199	0.00758
Manual	0.7336	0.738	0.3331	0.4196	0.8948	0.7324	0.9707
Average	26	26	26	26	26	25	26

(Table B.16 continued)

TASKTIME							
Loop-Carried							
Question:	6.11a	6.11b	6.12a	6.13a	6.13b	6.14a	6.14b
Table Cells Key: Pearson Correlation R / Prob > R / N obs.	# of informal courses in programming	Perceived Value of Informal Learning	Perceived usefulness of "other" experiences in programming	Percentage of Time in Non-Programming Work	Percentage of Time in Seq. Programming Work	# of Programming Languages used at any time	# of Programming Languages used within last 6 months
Variable=	QNINFRML	QVINFRML	QGUSEFUL	QNPGPCT	QSPGPCT	QALLLANG	QACTLANG
Overall	**0.40315**	0.50128	-0.12027	-0.20839	0.20921	-0.19398	-0.13893
Average	**0.0411**	0.2518	0.5669	0.3175	0.3155	0.3424	0.5078
	26	7	25	25	25	26	25
Annotated	**0.45864**	0.54739	-0.11358	-0.11532	0.1302	-0.20896	-0.10771
Average	**0.0184**	0.2034	0.5888	0.5831	0.5351	0.3056	0.6083
	26	7	25	25	25	26	25
Manual	0.28021	0.27817	-0.15266	-0.29491	0.28015	-0.20031	-0.21946
Average	0.1656	0.5458	0.4663	0.1524	0.175	0.3265	0.2919
	26	7	25	25	25	26	25
Anti-	**0.50828**	0.52034	0.01479	-0.00224	0.05426	-0.31605	-0.08104
dependence	**0.008**	0.2312	0.9441	0.9915	0.7967	0.1157	0.7002
Average	**26**	7	25	25	25	26	25
Flow	0.25614	0.03865	-0.12617	-0.33521	0.3082	0.15311	0.01144
dependence	0.2066	0.9344	0.5479	0.1014	0.1339	0.4552	0.9567
Average	26	7	25	25	25	26	25
Output	0.18574	0.23253	-0.17321	-0.17971	0.14385	-0.30462	-0.26266
dependence	0.3636	0.6158	0.4077	0.39	0.4927	0.1303	0.2046
Average	26	7	25	25	25	26	25
Anti-dep.	**0.65866**	0.43755	-0.00695	0.11294	-0.057	**-0.34308**	-0.06661
Annotated	**0.0003**	0.3262	0.9743	0.5993	0.7914	**0.0932**	0.7571
Average	**25**	7	24	24	24	**25**	24
Anti-dep.	0.18018	0.47187	0.0407	-0.08793	0.12241	-0.18981	-0.03659
Manual	0.3888	0.2851	0.8502	0.6829	0.5688	0.3635	0.8652
Average	25	7	24	24	24	25	24
Flow dep.	0.20157	0.23499	-0.21615	-0.19472	0.18564	0.11189	0.00964
Annotated	0.3234	0.612	0.2994	0.351	0.3743	0.5863	0.9635
Average	26	7	25	25	25	26	25
Flow dep.	0.16946	-0.41832	0.09745	-0.29732	0.25973	0.18186	0.08155
Manual	0.4181	0.3503	0.6506	0.1583	0.2203	0.3843	0.7048
Average	25	7	24	24	24	25	24
Output dep.	0.06819	0.10768	-0.04381	-0.13972	0.10885	-0.2646	-0.20958
Annotated	0.7407	0.8183	0.8353	0.5053	0.6045	0.1915	0.3147
Average	26	7	25	25	25	26	25
Output dep.	0.26829	0.29341	-0.32966	-0.16573	0.13248	-0.3097	-0.28837
Manual	0.1851	0.5231	0.1076	0.4285	0.5279	0.1236	0.1621
Average	26	7	25	25	25	26	25

(Table B.16 continued)

TASKTIME							
Loop-Carried							
Question:	6.15a	6.16a	6.17a	6.18a	6.18b	6.18d	6.18f
Table Cells Key: Pearson Correlation R Prob > R N obs.	Perceived Value of Manual Method	Perceived Value of Annotated Method	Enjoyment of working problems	Frequency of use of: IF statements	Frequency of use of: Sequential DO	Frequency of use of: CASE statement	Frequency of use of: RECURSION
Variable=	QVMANMTH	QVANNMTH	QENJOYED	QIFF	QSEQDOF	QCASEF	QRECURSE
Overall	-0.0107	-0.03044	0.08193	0.00737	0.1092	0.03856	-0.19464
Average	0.9586	0.8827	0.6907	0.9715	0.5954	0.8516	0.3407
	26	26	26	26	26	26	26
Annotated	-0.08824	-0.0047	0.09141	0.02658	0.00671	0.02213	-0.06609
Average	0.6682	0.9818	0.657	0.8975	0.974	0.9146	0.7484
	26	26	26	26	26	26	26
Manual	0.05057	-0.08784	0.08512	-0.05481	0.27311	0.06115	**-0.36134**
Average	0.8062	0.6696	0.6793	0.7903	0.177	0.7667	**0.0697**
	26	26	26	26	26	26	**26**
Anti-	0.08531	0.20329	0.09159	-0.02799	0.06022	0.27847	0.19029
dependence	0.6786	0.3192	0.6563	0.892	0.7701	0.1684	0.3518
Average	26	26	26	26	26	26	26
Flow	-0.00978	-0.2598	-0.13269	0.0306	0.21202	-0.15462	**-0.51558**
dependence	0.9622	0.1999	0.5182	0.882	0.2984	0.4507	**0.007**
Average	26	26	26	26	26	26	**26**
Output	-0.138	-0.02076	0.24166	0.04731	-0.04692	-0.05978	-0.12534
dependence	0.5014	0.9198	0.2343	0.8185	0.82	0.7718	0.5418
Average	26	26	26	26	26	26	26
Anti-dep.	-0.0091	0.12765	0.204	0.0512	-0.01738	0.30846	0.27042
Annotated	0.9655	0.5432	0.328	0.808	0.9343	0.1336	0.1911
Average	25	25	25	25	25	25	25
Anti-dep.	0.19421	0.26109	-0.08509	-0.1347	0.06859	0.12369	0.13617
Manual	0.3523	0.2074	0.6859	0.5209	0.7446	0.5558	0.5163
Average	25	25	25	25	25	25	25
Flow dep.	-0.17674	-0.25815	-0.16674	-0.14504	0.00887	**-0.43104**	-0.31112
Annotated	0.3878	0.2029	0.4156	0.4796	0.9657	**0.0279**	0.1219
Average	26	26	26	26	26	**26**	26
Flow dep.	0.18583	-0.13092	-0.05638	0.2128	0.2744	0.21112	**-0.46913**
Manual	0.3738	0.5328	0.789	0.3071	0.1844	0.311	**0.018**
Average	25	25	25	25	25	25	**25**
Output dep.	-0.01731	0.12048	0.21347	0.16732	-0.0164	0.15844	-0.07384
Annotated	0.9331	0.5577	0.2951	0.4139	0.9366	0.4395	0.72
Average	26	26	26	26	26	26	26
Output dep.	-0.24658	-0.16308	0.22867	-0.13366	-0.02913	-0.31094	-0.21088
Manual	0.2246	0.426	0.2612	0.5151	0.8877	0.1221	0.3011
Average	26	26	26	26	26	26	26

(Table B.16 ends)

B.5 Summary Subject Performance Statistics

These are statistical summaries of subject overall performance, such as the length of time required to complete various sections of the hypertext document, the number of keyboard shortcuts used, *etc.* Because the experimental materials differed between the two training groups, the form of presentation is similar to that of Appendix B.3: a introduction with the variable NAME indicated with its explanation, and since all values are continuous, a table comparing the TRAINING groups separately and combined, with number of responses received (N), mean, standard deviation, minimum, median, and maximum values, together with the p probability of the associated Wilcoxon test. Note that not all statistics were collected for four subjects.

Number of **Xbro***wser* runs to complete experiment: NRUNS

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
NRUNS	AF:	13	9.2	1.7	7	9	13	
	DA:	9	9.4	1.4	7	10	12	
	BOTH:	22	9.3	1.5	7	9	13	0.56

Number of edit operations completed with keyboard shortcuts: NSHORT
Number of edit operations completed with mouse: NEDITS
Number of edit operations, total: NEDTOT
Percentage of edit operations completed with keyboard: EKEYPCT

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
NSHORT	AF:	16	211	166.4	0	288.0	499	
	DA:	10	163	171.7	0	128.0	378	
	BOTH:	26	192	166.7	0	252.5	499	0.43
NEDITS	AF:	16	376	60.3	251	362.0	548	
	DA:	10	379	33.8	341	367.0	457	
	BOTH:	26	377	50.9	251	363.5	548	0.60
NEDTOT	AF:	16	586	195.8	352	626.5	1047	
	DA:	10	541	176.3	341	533.0	774	
	BOTH:	26	569	186.2	341	587.0	1047	0.69
EKEYPCT	AF:	16	30%	21%	0%	42%	56%	
	DA:	10	23%	24%	0%	22%	49%	
	BOTH:	26	28%	22%	0%	40%	56%	0.51

Number of document buttons activated with mouse: NBUTTONS
Number of document buttons activated with keyboard shortcuts: NKEYS
Number of document button events, total: NBUTTOT
Percentage of buttons activated with keyboard: BKEYPCT

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
NBUTTONS	AF:	13	406	120.7	265	394.0	639	
	DA:	9	441	133.2	223	467.0	600	
	BOTH:	22	420	124.1	223	436.5	639	0.55
NKEYS	AF:	13	119	108.4	0	114.0	275	
	DA:	9	97	90.7	1	61.0	286	
	BOTH:	22	110	99.9	0	106.0	286	0.87
NBUTTOT	AF:	13	525	65.1	431	508.0	640	
	DA:	9	538	87.0	373	543.0	661	
	BOTH:	22	530	73.1	373	521.5	661	0.42
BKEYPCT	AF:	13	23%	20%	0%	22%	48%	
	DA:	9	19%	18%	0%	9%	52%	
	BOTH:	22	21%	19%	0%	22%	52%	0.95

Total number of editor and document button keyboard shortcuts: NSTROKE
Total number of editor and document button mouse interactions: NINTRCT
Total number of editor and document button interactions: NTOTAL
Percentage of interactions using keyboard shortcuts: TKEYPCT

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
NSTROKE	AF:	13	343	212.0	0	340.0	598	
	DA:	9	278	204.2	3	369.0	616	
	BOTH:	22	316	206.5	0	354.5	616	0.59
NINTRCT	AF:	13	786	161.2	561	788.0	1187	
	DA:	9	824	145.7	587	863.0	980	
	BOTH:	22	801	152.6	561	814.0	1187	0.39
NTOTAL	AF:	13	1129	256.0	788	1142	1687	
	DA:	9	1101	231.1	737	1210	1368	
	BOTH:	22	1117	240.8	737	1176	1687	0.92
TKEYPCT	AF:	13	28%	16%	0%	28%	49%	
	DA:	9	23%	15%	0%	27%	50%	
	BOTH:	22	26%	16%	0%	28%	50%	0.64

Time to complete (or time spent in) hypertext document sections

LEARNING PORTION:

Introduction to **Xbro***wser*: TXBINTRO

Table of Contents: TCONTENT

Section 1: Introduction (to experiment): TINTRO

Section 2: P-F Programming Language: TPF

Section 3.1: Program Comprehension Techniques (shared) TTECHNIX

Sections 3.2–3.5: *METHOD*[0], and Examples: TMETHOD

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
TXBINTRO	AF:	16	359	295.0	99	295.5	1326	
	DA:	10	334	201.1	64	329.5	697	
	BOTH:	26	349	258.7	64	299.5	1326	0.85
TCONTENT	AF:	15	132	115.1	24	89.0	390	
	DA:	10	319	524.4	21	172.0	1784	
	BOTH:	25	207	345.9	21	110.0	1784	0.32
TINTRO	AF:	16	119	39.6	71	110.0	207	
	DA:	10	115	34.3	50	110.0	167	
	BOTH:	26	117	3.07	50	110.0	207	0.87
TPF	AF:	16	634	243.3	329	599.0	1321	
	DA:	10	610	232.6	315	587.5	1141	
	BOTH:	26	625	234.8	315	599.0	1321	0.73
TTECHNIX	AF:	16	542	233.8	335	474.0	1259	
	DA:	10	570	331.8	266	419.5	1415	
	BOTH:	26	553	269.5	266	466.5	1415	0.65
TMETHOD	AF:	16	2195	526.7	1389	2137.5	3483	
	DA:	10	2826	1487.4	1552	2235.5	6116	
	BOTH:	26	2438	1029.9	1389	2137.5	6116	0.51

[0]Where *METHOD* is Algebraic Formulation OR Dependence Analysis

Time to complete (or time spent in) hypertext document sections

RESEARCH PORTION:

Section 4: Review (working problems + correcting mistakes): RVIEWTOT
Section 5: Problems (all parts of all regular problems): PROBTOT
Section 6: Programming Background Questionnaire: TQUEST
Section 7: Debriefing, review of answers to problems, *etc.*: TPOST

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
RVIEWTOT	AF:	16	781	262.0	455	691.0	1263	
	DA:	10	841	236.1	462	842.0	1258	
	BOTH:	26	804	249.2	455	739.0	1263	0.44
PROBTOT	AF:	16	5390	731.1	3970	5560.0	6818	
	DA:	10	6043	995.4	4594	5804.5	7586	
	BOTH:	26	5641	884.6	3970	5599.0	7586	0.16
TQUEST	AF:	16	840	379.9	526	672.5	1714	
	DA:	10	768	329.6	402	680.5	1451	
	BOTH:	26	812	356.3	402	679.5	1714	0.81
TPOST	AF:	16	492	454	58	325.0	1401	
	DA:	9	367	226.8	105	288.0	781	
	BOTH:	25	447	386.9	58	310.0	1401	0.93

The Review and Problems sections also break down into interesting components.

Review breakdown

Section 4a: Review (working problems): TREVIEW
Section 4b: Review (reviewing, and correcting any mistakes): TRRVIEW

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
TREVIEW	AF:	16	331	152.6	94	315.5	653	
	DA:	10	236	83.5	102	222.5	393	
	BOTH:	26	294	136.8	94	250.0	653	0.13
TRRVIEW	AF:	16	451	190.6	212	430.5	832	
	DA:	10	605	248.2	251	581.0	1048	
	BOTH:	26	510	223.3	212	462.0	1048	0.11

Problems breakdown

Section 5a: Problems, introductory: TPROBS

Section 5b: Problems, Comparison Displays (24×): TCOMPARE

Section 5c: Problems, Checklist Displays (24×): TCHCKLST

Section 5d: Problems, resting between (23×): TREST

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
TPROBS	AF:	16	93	72.2	20	79.0	306	
	DA:	10	68	32	19	72.5	132	
	BOTH:	26	83	60.5	19	73.5	306	0.46
TCOMPARE	AF:	16	428	149.0	265	378.0	870	
	DA:	10	409	65.9	288	419.5	490	
	BOTH:	26	421	122.4	265	402.5	870	0.69
TCHCKLST	AF:	16	4961	649.9	3705	5059.5	5948	
	DA:	10	5634	974.9	4208	5478.5	7166	
	BOTH:	26	5220	840.8	3705	5175.5	7166	0.12
TREST	AF:	16	500	110.2	355	476.0	713	
	DA:	10	553	190.6	365	500.5	1033	
	BOTH:	26	520	145.1	355	476.0	1033	0.71

Time spent in experiment up to Debriefing: MAINTOT

Time spent in experiment including Debriefing: EXPTOT

NAME	group	N	mean	sd	min	med	max	Wilcoxon p
MAINTOT	AF:	15	11366	1976	8962	10482	17013	
	DA:	10	13046	2814	10263	12935	20157	
	BOTH:	25	12038	2440	8962	11364	20157	0.09
EXPTOT	AF:	15	11880	2244	9833	11062	18414	
	DA:	9	13341	2896	10849	13045	20300	
	BOTH:	24	12428	2551	9833	11621	20300	0.07

Bibliography

[AC84] A. A. Afifi and Virginia Clark. *Computer-Aided Multivariate Analysis.* Lifetime Learning, Belmont, CA, 1984.

[AK87] R. Allen and K. Kennedy. Automatic translation of fortran programs to vector form. *ACM Transactions on Programming Languages and Systems*, 9(4):491–542, Oct. 1987.

[BBMS92] Albert Badre, Margaret Beranek, J. Morgan Morris, and John Stasko. Assessing program visualization systems as instructional aids. In Ivan Tomek, editor, *Computer assisted learning : 4th International Conference, ICCAL '92*, Lecture notes in computer science, No. 602, pages 87–99. Springer-Verlag, Berlin, 1992.

[BD88] Deborah A. Boehm-Davis. Software comprehension. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, chapter 5, pages 107–121. North-Holland, Amsterdam, 1988.

[Ber66] A. J. Bernstein. Analysis of programs for parallel processing. *IEEE Transactions on Electronic Computers*, 15(5):757–763, Oct. 1966.

[BGA90] Walter S. Brainerd, Charles H. Goldberg, and Jeanne C. Adams. *Programmer's Guide to FORTRAN 90.* McGraw-Hill, New York, 1990.

[BLCG92] Tim J. Berners-Lee, Robert Cailliau, and Jean-François Groff. The world-wide web. *Computer Networks and ISDN Systems*, 25(4-5):454–459, 1992.

[Bro80] Ruven E. Brooks. Studying programmer behavior experimentally: The problem of proper methodology. *Communications of the ACM*, 23(4):207–213, April 1980.

[Bus45] Vannevar Bush. As we may think. *Atlantic Monthly*, pages 106–107, July 1945.

[CCH+88a] David Callahan, Keith D. Cooper, Robert T. Hood, Ken Kennedy, and Linda Torczon. ParaScope: A parallel-programming environment. Tech. Report Comp. TR88-77, Rice University, Houston, TX, Sept. 1988. (another version of [CCH+88b]).

[CCH+88b] David Callahan, Keith D. Cooper, Robert T. Hood, Ken Kennedy, and Linda Torczon. ParaScope: A parallel-programming environment. *International Journal of Supercomputer Applications*, 2(4):84–99, Winter 1988. (another version of [CCH+88a]).

[Che93] Doreen Cheng. A survey of parallel programming languages and tools. Technical Report RND-03-005, NASA Ames Research Center, Moffett Field, CA, March 1993.

[CMN80] Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):396–410, July 1980.

[CMN83] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, Hillsdale, NJ, 1983.

[Coh88] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences.* Lawrence Erlbaum, Hillsdale, NJ, second edition, 1988.

[Con86] Jeff Conklin. A survey of hypertext. Technical Report STP-356-86, MCC Software Technical Report, Austin, TX, Oct. 1986.

[Cra87] Gregory Crane. From the old to the new: Integrating hypertext into traditional scholarship. In *Hypertext '87 proceedings*, pages 51–56. ACM, New York, Nov. 13–15, 1987.

[Cur88] Bill Curtis. Five paradigms in the psychology of programming. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, chapter 4, pages 87–105. North-Holland, Amsterdam, 1988.

[Ega88] Dennis E. Egan. Individual differences in human-computer interaction. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, chapter 24, pages 543–568. North-Holland, Amsterdam, 1988.

[ES93] K. Anders Ericsson and Herbert A. Simon. *Protocol Analysis*. MIT Press, Cambridge, MA, revised edition, 1993.

[Fis87] Carolanne Fisher. Advancing the study of programming with computer-aided protocol analysis. In Gary M. Olson, Sylvia Sheppard, and Elliot Soloway, editors, *Empirical Studies of Programmers: Second Workshop*, pages 198–216. Ablex, Norwood, NJ, 1987.

[Fis91] Carolanne Fisher. *Protocol Analyst's Workbench: Design and Evaluation of Computer-Aided Protocol Analysis*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1991.

[Fre87] Daniel H. Freeman Jr. *Applied Categorical Data Analysis*. Statistics: Textbooks and Monographs, No. 79. Marcel Dekker, New York, NY, 1987.

[GC92] Jean D. Gibbons and Subhabrata Chakraborti. *Nonparametric Statistical Inference*. Statistics: Textbooks and Monographs, No. 131. Marcel Dekker, New York, NY, third edition, 1992.

[GC93] Raghava G. Gowda and Donald R. Chand. Exploration of the impact of individual and group factors on programmer productivity. In *Proceedings of the 21st Annual ACM Computer Science Conference*, pages 338–345, New York, NY, 1993. ACM.

[GD74] J. D. Gould and P. Drongowski. An exploratory study of computer program debugging. *Human Factors*, 16:258–277, 1974.

[GS89] Raghava G. Gowda and Stanley Saxton. Study of factors influencing the productivity of programming teams. In *Proceedings of the 17th Annual ACM Computer Science Conference*, page 484, New York, NY, 1989. ACM. (extended abstract).

[GSM86] N. L. Garrett, K. E. Smith, and N. Meyrowitz. Intermedia: Issues, strategies, and tactics in the design of a hypermedia document system. In *CSCW : Proceedings of the Conference on Computer-Supported Cooperative Work*, New York, NY, Dec. 1986. ACM. (Held in Austin, Texas).

[GT90] Dirk Grunwald and Chris Torek. *SeeTeX Installation Guide*. SeeTeX package version 2.18.5, 1990.

[Hag90] Mohammad Reza Haghighat. Symbolic dependence analysis for high performance parallelizing compilers. Master's thesis, University of Illinois, Center for Supercomputing Research and Development, May 1990.

[Ham84] John M. Hammer. Statistical methodology in the literature on human factors in computer programming. In Gavriel Salvendy, editor, *Human-Computer Interaction*, pages 189–193. Elsevier Science, Amsterdam, 1984.

[HSA89] Matthew E. Hodges, Russell M. Sasnett, and Mark S. Ackerman. A construction set for multimedia applications. *IEEE Software*, 6(1):37–43, Jan. 1989.

[Ise88] Errol R. Iselin. Conditional statements, looping constructs, and program comprehension: an experimental study. *International Journal of Man-Machine Studies*, 28(1):45–66, Jan. 1988.

[KA86] Claudius M. Kessler and John R. Anderson. Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, 2(2):135–166, 1986.

[KKP+81] D. J. Kuck, R. H. Kuhn, D. A. Padua, B. Leasure, and M. Wolfe. Dependence graphs and compiler optimizations. In *Conference Record of the Eight Annual ACM Symposium on Principles of Programming Languages*, pages 207–218. ACM, Jan. 1981.

[KL90] Bogdan Korel and Janusz Laski. Dynamic slicing of computer programs. *Journal of Systems and Software*, 13(3):187–195, 1990.

[KLS85] Richard J. Koubek, William K. LeBold, and Gavriel Salvendy. Predicting performance in computer programming courses. *Behaviour and Information Technology*, 4(2):113–129, 1985.

[Knu79] Donald E. Knuth. TEX *and* METAFONT*: new directions in typesetting.* Digital Press, Bedford, MA, 1979.

[Lam94] Leslie Lamport. LaTeX*: a document preparation system.* Addison-Wesley, Reading, MA, second edition, 1994.

[Lan88] Thomas K. Landauer. Research methods in human-computer interaction. In Martin Helander, editor, *Handbook of Human-Computer Interaction*, chapter 42, pages 905–928. North-Holland, Amsterdam, 1988.

[Las90] Janusz Laski. Data flow testing in STAD. *Journal of Systems and Software*, 12(1):3–14, 1990.

[LD92] Brigitte de La Passardiere and Aude Dufresne. Adaptive navigational tools for educational hypermedia. In Ivan Tomek, editor, *Computer assisted learning : 4th International Conference, ICCAL '92*, Lecture notes in computer science, No. 602, pages 555–567. Springer-Verlag, Berlin, 1992.

[LPB87] E. Lueke, P. D. Pagery, and C. R. Brown. User requirement gathering through verbal protocol analysis. In Gavriel Salvendy, editor, *Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems*, pages 135–157. Elsevier, Amsterdam, 1987.

[LRM92] Jun Li, Allen Rovick, and Joel Michael. ABASE: A hypermedia-based tutoring and authoring system. In Ivan Tomek, editor, *Computer assisted learning : 4th International Conference, ICCAL '92*, Lecture notes in computer science, No. 602, pages 380–390. Springer-Verlag, Berlin, 1992.

[Lyl84] James Robert Lyle. *Evaluating Variations on Program Slicing for Debugging.* PhD thesis, Computer Science Department, University of Maryland, December 1984.

[Mac89] Wendy E. Mackay. EVA: An experimental video annotator for symbolic analysis of video data. *SIGCHI Bulletin*, 21(2):68–71, Oct. 1989.

[Mar92] Brian D. Markey. HyTime and MHEG. In *37th Annual IEEE International Computer Conference: COMPCON SPRING '92*, pages 25–40, Piscataway, NJ, 1992. IEEE.

[McC86] Robert B. McCall. *Fundamental Statistics for Behavioral Sciences.* Harcourt Brace Jovanovich, fourth edition, 1986.

[ME92] Tomasz Müldner and Mohammed Elammari. OBJECTOR: Yet another authoring system. In Ivan Tomek, editor, *Computer assisted learning : 4th International Conference, ICCAL '92*, Lecture notes in computer science, No. 602, pages 478–490. Springer-Verlag, Berlin, 1992.

[Mye90] Brad A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, 1(1):97–123, March 1990. (cited in [BBMS92]).

[Nel81] Ted Nelson. *Literary Machines.* Available from the author, Box 128, Swarthmore, PA 19081, 1981. (cited in [TI87]).

[OCN89] Paul W. Oman, Curtis R. Cook, and Murthi Nanja. Effects of programming experience in debugging semantic errors. *Journal of Systems and Software*, 9(3):197–207, March 1989.

[Par50] Mildred Parten. *Surveys, Polls, and Samples: Practical Procedures.* Harper & Brothers, 1950.

[Pen82] Nancy Pennington. Cognitive component of expertise in computer programming: a review of the literature. Technical Report No. 46, University of Michigan, 1982. (cited in [Ise88]).

[Ras87] Jef Raskin. The hype in hypertext: A critique. In *Hypertext '87 proceedings*, pages 325–330. ACM, New York, Nov. 13–15, 1987.

[SAS90a] SAS Institute Inc. *SAS Procedures Guide.* SAS Institute, Inc., Cary, NC, version 6 edition, 1990.

[SAS90b] SAS Institute Inc. *SAS User's Guide: Statistics.* SAS Institute, Inc., Cary, NC, version 6 edition, 1990.

[SBE83] Elliot Soloway, Jeffrey Bonar, and Kate Ehrlich. Cognitive strategies and looping constructs: An empirical study. *Communications of the ACM*, 26(11):853–860, Nov. 1983.

[Sch90] Herbert Schildt. *C, the complete reference.* Osborne McGraw-Hill, Berkeley, CA, second edition, 1990.

[She81] B. A. Sheil. The psychological study of programming. *ACM Computing Surveys*, 13(1):101–120, 1981.

[Shn80] Ben Shneiderman. *Software Psychology*. Little, Brown, Boston, 1980.

[SJS89] P. Sanderson, J. M. James, and K. S. Seidler. SHAPA: An interactive software environment for protocol analysis. *Ergonomics*, 32:1271–1302, 1989.

[SLY89] Zhiyu Shen, Zhiyuan Li, and Pen-Chung Yew. An empirical study on array subscripts and data dependences. In *Proceedings of the 1989 International Conference on Parallel Processing*, volume II, pages 145–152, St. Charles, IL, Aug. 1989.

[SP91] John T. Stasko and Charles Patterson. Understanding and characterizing program visualization systems. Technical Report GIT-GVU-01/17, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Sept. 1991.

[SSK92] John B. Smith, Dana Kay Smith, and Eileen Kupstas. Machine-recorded protocols: Tools and techniques. In *Proceedings of the Human Factors Society 36th Annual Meeting*, pages 369–373, Santa Monica, CA, 1992.

[Sta90] John T. Stasko. TANGO: A framework and system for algorithm animation. *Computer*, 23(9):27–39, Sept. 1990.

[Str91] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, second edition, 1991.

[SW93] M. Samadzadeh and W. Wichaipanitch. An interactive debugging tool for c based on dynamic slicing and dicing. In *Proceedings of the 21st Annual ACM Computer Science Conference*, pages 30–37, New York, NY, 1993. ACM.

[SWCB93] Vincent Shah, Ray Waddington, Tom Carey, and Peter Buhr. The recognition of concurrent programming plans by novice and expert programmers: Implications for the parsimony of the plan theory of programming experts. In Curtis R. Cook, Jean C. Scholtz, and James C. Spohrer, editors, *Empirical Studies of Programmers: Fifth Workshop*, page 229. Ablex, Norwood, NJ, 1993. (abstract).

[TI87] Randall H. Trigg and Peggy M. Irish. Hypertext habitats: Experiences of writers in NoteCards. In *Hypertext '87 proceedings*, pages 89–108. ACM, New York, Nov. 13–15, 1987.

[Tri89] Randall H. Trigg. Computer support for transcribing recorded activity. *SIGCHI Bulletin*, 21(2):72–74, Oct. 1989.

[Wad93] Ray Waddington. Concurrent microlanguages: Demonstration of an experimental method for the empirical study of concurrent programming. In Curtis R. Cook, Jean C. Scholtz, and James C. Spohrer, editors, *Empirical Studies of Programmers: Fifth Workshop*, page 231. Ablex, Norwood, NJ, 1993. (abstract).

[WB85] Judith D. Wilson and Gerald F. Braun. Psychological differences in university computer student populations. *SIGCSE Bulletin*, 17(1):166–177, March 1985.

[Wei79] Mark Weiser. *Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method.* PhD thesis, University of Michigan, Ann Arbor, 1979.

[Wei82] Mark Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(5):446–452, July 1982.

[Wei84] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, July 1984.

[WHN93] Judith D. Wilson, Nathan Hoskin, and John T. Nosek. Benefits of collaboration for student programmers. *SIGCSE Bulletin*, 25(1):160–164, March 1993.

[Win71] B. J. Winer. *Statistical Principles in Experimental Design.* McGraw-Hill, second edition, 1971.

[WL86] Mark Weiser and Jim Lyle. Experiments on slicing-based debugging aids. In Elliot Soloway and Sitharama Iyengar, editors, *Empirical Studies of Programmers: papers presented at the First Workshop on Empirical Studies of Programmers*, pages 187–197. Ablex, Norwood, NJ, 1986.

[WN71] D. A. Waterman and A. Newell. Protocol analysis as a task for artificial intelligence. *Artificial Intelligence*, 2:285–318, 1971. (cited in [SSK92, ES93, Fis87]).

[WN73] D. A. Waterman and A. Newell. PAS-II: An interactive task-free version of an automatic protocol analysis system. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 431–445, Menlo Park, CA, 1973. Stanford Research Institute. (cited in [SSK92, ES93, Fis87]).

[YHMD88] Nicole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer*, 21(1):81–96, Jan. 1988.

[ZR93] Min Zheng and Roy Rada. Shyd – a model for bridging text and hypermedia. In *Proceedings of the 21st Annual ACM Computer Science Conference*, pages 418–424, New York, NY, 1993. ACM.