
Investigations into History Tools for User Support

Alison Lee

Technical Report CSRI-271
April 1992

Computer Systems Research Institute
University of Toronto
Toronto, Canada
M5S 1A1

The Computer Systems Research Institute (CSRI) is an interdisciplinary group formed to conduct research and development relevant to computer systems and their application. It is an Institute within the Faculty of Applied Science and Engineering and the Faculty of Arts and Science, at the University of Toronto, and is supported in part by the Natural Sciences and Engineering Research Council of Canada



**Investigations into
History Tools for User Support**

Alison Lee

A thesis submitted in conformity with the requirements
for the Degree of Doctor of Philosophy,
Graduate Department of Computer Science, at the
University of Toronto

© Copyright 1992 by Alison Lee

**In the Memory of
My Grandparents**

A man, viewed as a behaving system,
is quite simple. The apparent complexity
of his behaviour over time is largely
a reflection of the complexity of the environment
in which he finds himself.

Herbert A. Simon,
The Science of the Artificial,
MIT Press (1969).

Table of Contents

Table of Contents	i
Abstract	vii
For The Record	ix
Acknowledgements	xi
Chapter 1: Introduction	1
Motivation	2
Considerations in the Design of User Support	2
Users are Individuals	2
User Behaviours are Variable	3
User-Computer Interactions are Complex and Difficult	4
Implications for User Support Endeavours	5
History Tools : A Proposed User-Support Candidate	5
User Support Capabilities Rely on History Information	6
Humans Draw Heavily from Past Experiences	6
History-Cued Problem Solving Facilitates Learning	6
History Information Helps Users Cope with Improvisation	7
History Tools Relieve Cognitive and Physical Burdens	8
Problem Statement	8
Dissertation Roadmap	8
Chapter 2: Uses of Information in a History	11
Taxonomy	11
History for Reuse	13
Reuse a Script Literally	13
Reuse an Interaction with Possible Modifications	14
Reuse Operations as a Functional Group	15
History for Inter-referential I/O	18

History for Error Recovery	18
Command Correction	19
System Recovery	19
History for Navigation	20
History for Reminding	20
User Consultation	20
Visual Cues	22
History for User Modelling	23
History for User Interface Adaptation	23
Assessment	23
Results and Discussion	24
Concluding Remarks	26
Chapter 3: Cognitive and Behavioural Issues	27
Contents of a User History	27
Actions and Objects Referenced in Actions	28
User Inputs and System Responses	28
States As Well As Transitions	29
Specific Information and Peripheral Information	29
Externalized Information and Internalized Information	30
User and Behavioural Information	31
Basic History Components	31
Collection of History	32
Selective Logs	32
User Collection Tools	32
Presentation of History	33
Spontaneous Reminding	33
Deliberate Reminding	34
Visual Scan and/or Search	34
Problem Solving and Improvisations	34
Submission of History Item	35
Identification	35
Modification	36
Retraction	36
Coordination	36
Administration of History	37
Separating Independent Histories	37
Accessing History Information	38
Aging, Saving and Discarding History Information	38
Concluding Remarks	39

Chapter 4: User Interaction Behaviour and History Usage	41
Behavioural Studies of User Interactions and History Use	41
Exploratory Study of User Interactions and History Usage	42
Method	43
UNIX Shells and History Tools	43
Data Collection	44
Subjects	45
Procedure	45
Results	45
Three Behavioural Patterns	46
History–Tool Functions Used by csh Subjects	46
History–Tool Functions Used by tcsh Subjects	48
Discussion	48
Concluding Remarks	49
Chapter 5: Locality in User Interactions	51
Program Memory Reference Research	51
Working Sets	52
Locality	52
Relevance to the Study of Recurrences	54
Previous HCI Attempts to Apply these Concepts	56
Importance of Considering Locality	56
Users' Session Traces	58
Study 1 : Does Locality Exist in User Interactions?	58
Results and Discussion	59
Locality in Full Command Lines	59
Locality in Command Names Only	64
Study 2 : Does Locality Exist By Chance?	68
Statistical Test	68
Method	69
Results and Discussion	69
Study 3 : Does Locality Enhance the Recurrence Picture?	69
Study 3A: History–for–Reuse Occurs Within a Locality	69
Method	72
Results and Discussion	72
Study 3B: Locality and Prediction of Reuse Candidates	74
Method	74
Results and Discussion	74
Concluding Remarks	77
Chapter 6: User Effort in the Command Specification Task	79
User Effort Study	79

Analytic Methodology	80
User Effort	81
Command–Specification Task and Methods	81
Command Feature	82
History Menu	82
Step Buffer	83
Constraints of the Study	83
Estimates for Predictions	84
MHP Operators	84
Application–Specific Parameters	86
Procedure	87
Results and Discussion	90
Human Performance within a Class of History Designs	90
Human Performance between Classes of History Designs	91
Typing versus Using a History Tool	93
Accuracy of the Cognitive Models’ Predictions	95
Method	95
Subjects	96
Materials and Apparatus	96
Tasks	96
Procedure	97
Results and Discussion	98
Remarks about Our Studies	104
Problems with Cognitive Modelling	104
Future Work	105
Concluding Remarks	106
Chapter 7: Conclusion	109
Contributions	109
Future Directions	111
Appendix A: Selected Systems Supporting History Tools	113
Appendix B: Locality Detection Algorithm	115
Appendix C: Cognitive Modelling Framework	119
Model Human Processor	119
GOMS	121
Critical Elements of a Cognitive Model	121
Shortcomings of Engineering Models	123
Extensions to MHP and GOMS	123
GOMS Model of Immediate Behaviour	124
ARTLESS: An Algorithmic Response–Time Language	124

Cost Scheme for an ARTLESS Algorithm	126
Appendix D: Visual Search of a History Menu	127
Estimation of Visual Search Operator for a History Menu	127
Method	128
Subjects	128
Materials and Apparatus	129
Scenarios in Locality and Non-Locality Experiment	129
Procedure	130
Results and Discussion	131
Number of Errors	131
Search Times	133
Time Parameter for the Pure Menu Search Operator	135
Concluding Remarks	138
Appendix E: Task Descriptions for Proposed Designs	139
Appendix F: Task Descriptions for Experimental Tasks	151
Bibliography	157

Abstract

History tools allow users to access past interactions kept in a *history* and to incorporate them into the context of their current operations. Such tools appear in various forms in many of today's computing systems, but despite their prevalence, they have received little attention as user support tools. This dissertation investigates, through a series of studies, history-based, user support tools. The studies focus on three primary factors influencing the utility of history-based, user support tools: design of history tools, support of a behavioural phenomenon in user interactions, and mental and physical effort associated with using history tools.

Design of history tools strongly influences a user's perception of their utility. In surveying a wide collection of history tools, we identify seven independent uses of the information with no single history tool supporting all seven uses. Based on cognitive and behavioural considerations associated with the seven history uses, we propose several kinds of history information and history functions that need to be supported in new designs of history tools integrating all seven uses of history.

An exploratory study of the UNIX environment reveals that user interactions exhibit a behavioural phenomenon, nominally referred to as *locality*. This is the phenomenon where users repeatedly reference a small group of commands during extended intervals of their session. We apply two concepts from computer memory research (i.e., working sets and locality) to examine this behavioural artifact and to propose a strategy for predicting repetitive opportunities and candidates. Our studies reveal that users exhibit locality in only 31% of their sessions whereas users repeat individual commands in 75% of their sessions. We also found that history tool use occurs primarily in locality periods. Thus, history tools which localize their prediction opportunities to locality periods can predict effectively the reuse candidates.

Finally, the effort, mental and physical, associated with using a history tool to expedite repetitive commands can influence a user's decision to use history tools. We analyze the human-information-processing operations involved in the task of specifying a recurrent command for a given approach and design (assuming that the command is fully generated and resides in the user's working memory and that users exhibit expert, error-free task performance behaviour). We find that in most of the proposed history designs, users expend less physical effort at the expense of more mental effort. The increased mental effort can be alleviated by providing history tools which require simpler mental operations (e.g., working memory retrievals and perceptual processing). Also, we find that the typing

approach requires less mental effort at the expense of more physical effort. Finally, despite the overhead associated with switching to the use of history tools, users (with a typing speed of 55 wpm or less) do expend less overall effort to specify recurrent commands (which have been generated and appear in working memory) using history tools compared to typing from scratch.

The results of the three sets of studies provide insights into current history tools and point favourably towards the use of history tools for user support, especially history tools that support the reuse of previous commands, but additional research into history tool designs and usability factors is needed. Our studies demonstrate the importance of considering various psychological and behavioural factors and the importance of different grains of analysis.

For The Record

Earlier versions of the following chapters were published or presented in 5 conferences, 1 workshop, and 1 book.

Chapter 1 is an abridged version of the paper appearing in Lee (1992a).

Chapter 2 is a revised version of a paper appearing in Lee (1990b).

Chapter 3 is a substantially revised version of a paper appearing in Lee (1990a).

Chapter 4 is a revised version of a paper appearing in Lee and Lochovsky (1990a).

Chapter 5 is a revised version of a paper appearing in Lee and Lochovsky (1990b).

The experiment in the section entitled "Accuracy of the Cognitive Models' Predictions" in Chapter 6 is a longer version of a short talk appearing in Lee (1992b).

The visual search experiment in Appendix D is a longer version of a poster appearing in Lee and Lochovsky (1991).

Acknowledgements

I am indebted to a great number of people who have made this research experience an enjoyable, enriching, and enlightening one. I hope I got everyone and I apologize for any oversight.

Fred Lochovsky, my supervisor, has been extremely supportive of my choice of research and has provided encouragement and guidance throughout the process. His faith in my abilities to tackle this dissertation has given me confidence during many a difficult period. Marilyn Mantei, my internal appraiser, has been an enthusiastic supporter and invaluable educator and colleague. Her input and suggestions have helped to shape the work described in Chapters 5 and 6. I value the countless number of hours of discussion that we had and I will dearly miss them. Ron Baecker's constructive critique have substantially improved the clarity and overall presentation of the thesis. The other members of my committee – Joan Cherry from FLIS, John Mylopoulos, and Dave Wortman – have been extremely patient and remained interested throughout the long road to completion. Ian Witten arranged his travel plans in order to attend the examination as the external examiner. His interest and comments are appreciated.

My research has benefited from the input of many people. Scott Graham generously answered questions about the research on working sets. Rob Tibshrani and Ruth Croxford from Statistics provided much needed advice on statistical analysis. Abi Sellen assisted with experimental design and analysis. Avi Naiman introduced me to SPLUS and helped me develop programs. Michael Gelblum, Avi Naiman, Alan Rosenthal, Kevin Schlueter, Mark Tapia were always willing to listen and to provide comments. Saul Greenberg from the University of Calgary kindly shared the user trace data he collected. This data was invaluable in the studies I conducted in Chapter 5. Furthermore, his research has influenced my investigation of history tools. Early in my research, I was fortunate to have spent two summers at Xerox PARC. Bill Jackson and Ken Pier introduced me to several of their programming environments in which the idea of using history tools for user support was born. Dan Swinehart and Polle Zelleweger allowed me to explore the use of history tools which led to the work described in Chapters 2 and 3. Gary Hardock, Gord Kurtenbach, Avi Naiman, Kevin Schlueter, Ian Small, Mark Tapia commented on early drafts of the thesis. Kevin Schlueter deserves an extra note of thanks for persisting through several early versions and helped me to express my ideas appropriately. Finally, many individuals who shall remain anonymous volunteered as subjects for my three experiments. Without them, there would not have been a thesis.

I have enjoyed the spirit of collegiality that pervades the group of individuals in the DGP lab. Thanks to Ron Baecker, Bill Buxton, Eugene Fiume, Alain Fournier, and Marilyn Mantei for providing a truly dynamic and stimulating environment and for allowing me to work there.

I have formed many friendships during my time in the department. Aside from the people mentioned earlier, others include John Amanatides, John Buchanan, Elsa Campuzano, Jimmy Chui, Arvind Gupta, Ralph Hill, Iva Lu, Kelly Mawby, Tom Milligan, Moira Minoughan, Marc Ouellette, Mike Papper, Pierre Poulin, Maria Raso, and Carson Woo. Thanks to Pierre Poulin and Marc Ouellette for organizing softball and broomball. I worked off a lot of frustrations on the squash and racquetball courts with the help of Richard Cleve, Gary Haddock, Avi Naiman, and Blaine Price.

I gratefully acknowledge the financial support provided by the Department of Computer Science, the Ontario Graduate Studies fund, and the University of Toronto Open Fellowship. George Drettakis, Jeff Lee, Carson Schutze, Ian Small, and Dmitri Nastos in DGP, John DiMarco in DB & OIS, and Brian Down and Sandra Smith at CSRI have been very helpful in dealing with system administration problems and requests. Thanks to Anna Heron, Penny Marchand, Teresa Miao, and Kathy Yen for dealing with many administrative details.

Most importantly, my parents – Agnes and Anthony – and Lucia, Ignatius, and Andrew have provided lots of support, encouragement, and love.

Chapter 1

Introduction

The field of *user support* is concerned with how to support user-computer interactions. Past efforts to address this problem have taken one of three approaches: user-interface design, run-time support, and user instruction. User-interface design stresses the importance of designing usable interfaces because usability is an important user support concern. Run-time support focuses on the provision of run-time tools and resources, computer-based and non-computer-based, because user interactions are complex and difficult and user-interface issues are not always resolved at design time. Through run-time support, designers carry over their concern for users from user-interface design to the use of computer systems. A third approach involves developing methodologies for instructing and training users on the use of computer systems. The premise is that user-computer interactions are enhanced with user instructions.

In this thesis, we are concerned mainly with the provision of run-time support. Past efforts have concentrated on three aspects of user interactions: *user-centered*, *task-centered*, and *interaction-centered* views. The user-centered view is concerned with issues related to user intentions and actions and in particular, cognitive, behavioural, and social issues associated with user-computer interactions. The task-centered view is concerned with issues related to task analysis, task description, and task-to-tool mappings [Croft, 1984]. The interaction-centered view is concerned with issues related to needs and protocols for effective communication [Hayes, Ball, & Reddy, 1981; Nickerson, 1976; Taylor, 1988a, 1988b; Thimbleby, 1980]. While the distinction between interaction-centered and user-centered views and between interaction-centered and task-centered views may not be clear cut, the distinction between user-centered and task-centered views is clear. In the task-centered view, the task drives the design of an interface while in the user-centered view, user needs drive the design of an interface and interface needs drive the design of the rest of a system [Norman, 1986].

Since we cannot hope to address all of the various user support concerns, our discussion focuses on user-centered, computer-based, run-time user support. Furthermore, our treatment of this aspect of user support is biased towards cognitive and behavioural issues related to the provision of tools and resources for smoothly integrating computing with user tasks. However, this is not to suggest that other issues like implementation, social and organizational context of computing, theory of task domains, and training of users are unimportant.

This chapter presents our motivations for focusing on user-centered, computer-based, run-time user support. Then, we examine three important considerations for user support. They highlight the multiplicity of concerns and challenges that underlie attempts to provide user support. Finally, we propose that history, and tools based on history, can address many of these user support concerns and challenges. Several indications of the value of history tools are presented followed by a problem statement and brief overview of the dissertation.

Motivation

Computer users can encounter difficulties in interacting with a computer system and therefore the system must enable users to deal with and manage these difficulties as they arise [Brown & Newman, 1985; Quinn & Russell, 1986]. There are three reasons why a user-centered, computer-based, run-time support approach is important for dealing with such difficulties: diverse user knowledge and experience, task complexity, and interaction barriers.

First, support needs to be *user-centered* because users have diverse computer knowledge and experience. Thus, system designers need to provide user-centered tools and resources that will address individual user needs and concerns. Second, support needs to be *computer-based* because users perform complex computing tasks. Thus, system designers need to provide computer-based tools and resources that will allow them to focus on conceptual and difficult aspects of performing such tasks and to accomplish them easily [Card, 1989; Gasser, 1986; Rissland, 1984]. Finally, support needs to be *run-time* because the powerful and sophisticated tools in computer systems present formidable barriers to a user's understanding and use of these systems [Hayes, Ball, & Reddy, 1981; Winograd, 1973]. Thus, system designers need to provide run-time tools and resources to help users overcome the "communication" and "utility" barriers as they arise.

Considerations in the Design of User Support

In order to provide effective run-time assistance to users, designers need to be aware of and to understand users, their behaviours, and their interactions with computers. A literature review reveals three important considerations:

- Users are individuals.
- User behaviours are variable.
- User-computer interactions are complex and difficult.

Users are Individuals

Studies of user task performance and system usage consistently show that user behaviour in user-computer interactions is non-homogeneous [Boies, 1974; Card, Moran, & Newell, 1983; Carey, 1982; Egan, 1988; Greenberg & Witten, 1988a; Hanson, Kraut, & Farber, 1984; Potosnak, Hayes, Rosson, Schneider, & Whiteside, 1986]. User classification schemes have been proposed to illustrate

and to characterize individual differences and traits [Carey, 1982; Norman, 1984; Potosnak, Hayes, Rosson, Schneider, & Whiteside, 1986]. Each class represents an interpretation of user behaviour and a characterization of distinctive user differences (e.g., cognitive factors, preferences, frequency of system use). These classifications are not absolute as certain implicit generalizations and assumptions about individuals are made.

User Behaviours are Variable

Studies reveal that user behaviours vary from situation to situation. Such behavioural variability is supported by research into learning, human cognition, interaction gulfs, and engagement.

Learning research reveals that, with training and exposure to a task, a user's skill evolves from a *problem-solving* stage to a *procedural* stage and then to an *automatic* stage [Anderson, 1982; Card, Moran, & Newell, 1983; Rasmussen, 1983]. In unfamiliar stages of task performance, users with minimal task knowledge rely on heuristic strategies to search and navigate a problem space (i.e., problem solve). As user skills develop, operator sequences for moving from problem state to problem state are more predictable and are integrated and composed into debugged cognitive structures, known as *mental schema*. Mental schema knowledge allows a problem solver to recognize a problem state and its associated operators. It is encoded as procedural rules and then, with further refinement, human behaviour can be described as if these rules are collapsed into methods controlled by a single rule. As tasks become well-practiced, actions become automatic, and users are unable to describe the rules for controlling their task performance. Depending on which stage users are at, their behaviour varies from one task exposure to another.

Human information processing resources and capacities constrain human cognition [Card, Moran, & Newell, 1986; Lindsay & Norman, 1977]. Such constraints encourage an intuitive preference for mental processing strategies that lead to sequences of simple operations [Rasmussen, 1983]. Strategies may involve fitting task activities to system functions, optimizing for transfer of previous task results, and minimizing the need for new information. These constraints, coupled with different human information processing demands imposed by different situations, lead to variability in observed user behaviours.

Gulfs between a person's goals and knowledge and a system's requirements are qualified in terms of two distances: *semantic* and *articulatory* distances [Hutchins, Hollan, & Norman, 1986; Norman, 1986]. *Semantic distance* is the distance between how users think of their tasks and the semantics of the interface language. *Articulatory distance* is the distance between the semantics of the interface language and the physical form of the interface language. The less natural the relationship between the two ends of a distance, the less natural the formulation or articulation of a user intention or user action and the less natural the interpretation or evaluation of system semantics or system response. Hence, users have more problems and their behaviour is more variable.

Finally, *engagement* is the qualitative feeling of manipulating interface objects [Hutchins, Hollan, & Norman, 1986]. Variability in user behaviour arises depending on how users engage a system. There are situations where user actions

and objects may be described precisely with an abstraction (i.e., *indirect engagement*) rather than direct manipulation (i.e., *direct engagement*) or vice versa and as a result, differences in user behaviour are observed. Current interfaces do not effectively integrate both direct and indirect engagement.

User-Computer Interactions are Complex and Difficult

User-computer interactions do not always proceed in a straightforward manner. User interface and task complexities, individual differences, and behavioural variability often make on-going user interactions complex and difficult. Behavioural studies have shown that user interactions include four kinds of activities: multi-tasking, improvisation, repetition, and error correction.

Studies have found that user activities do not always proceed in a serial and uninterrupted fashion. Users often engage in a number of activities by either processing them concurrently or digressing and switching back to an activity [Bannon, Cypher, Greenspan, & Monty, 1983; Cypher, 1986; Greenberg & Witten, 1985; Henderson & Card, 1986a; Stephenson, 1973]. In switching amongst several parallel activities, users internalize complex sequences of parallel activities to run tasks sequentially on a computer. Factors such as limited resources and structural and functional mismatches between human and computer systems, extent of control of mental processing, nature of tasks (e.g., problematic, routine, well-defined), and human characteristics affect a user's ability to perform multiple activities [Cypher, 1986; Gasser, 1986; Lee, 1992a; Lindsay & Norman, 1977; Miyata & Norman, 1986; Norman & Shallice, 1986; Suchman, 1983].

According to Suchman (1987), user actions are *situated*: all activities take place in some specific on-going concrete situation that demands immediate attention. Real situations are typically rife with uncertainties, complexities, and contingencies. They arise for any number of reasons including misalignment between computing and work processes, work contingencies, system design restrictions, poor functionality, skill level, and nature of tasks [Gasser, 1986]. Users must continually evaluate and decide what to do next, utilizing resources from the immediate surroundings; they need to improvise [Agre & Chapman, 1990]. Thus, user actions may be systematic but they are not planned in the traditional sense.

Studies of computing and non-computing behaviours observed repetition in user activities. Greenberg and Witten (1988b), Hanson, Kraut, and Farber (1984) and Krishnamurthy (1987) reported that a few commands in a user session constitute a large percentage of the commands issued. Greenberg (1988b) also noted repetitive behaviours in the form of replaying favourite songs, looking up information in manuals several times, favouring certain physical tools or cooking recipes. Recurrent activities arise for several reasons: problematic tasks necessitate a trial and error process; tasks are routine and frequent (e.g., users invoke a window system every time they log in) or inherently repetitive (e.g., deleting files in different directories); and a number of tasks are performed using the same actions because less effort is required or poorly designed systems force users to do so.

Finally, users are fallible even in a well-designed computer system. Human error falls into two categories: *mistakes* and *slips* [Norman, 1981; Norman, 1983]. *Mistakes* are errors resulting from the formulation of wrong intentions while *slips* are errors resulting from the execution of inappropriate actions for an intention.

Mistakes include errors associated with faulty decision making and misunderstandings, misinterpretations, or misdiagnosis of situations while using a system [Norman, 1983]. Examples include mistaking a system state (i.e., mode errors) or specifying an incomplete or ambiguous intention (i.e., description errors). Slips are attributed to unintentional activation of actions, false activation of actions, or failure to trigger correctly selected actions.

Implications for User Support Endeavours

The preceding observations about users, their behaviour, and their interactions with computers highlight several important concerns for the provision of user support. No single mechanism is available to address all the concerns but there are three basic user support capabilities that are desirable and are discussed briefly in this section: *user models*, *adaptation*, and *facilitation* [Lee, 1992a].

A *user model* contains such information as a user's preferences, knowledge, characteristics, intentions, behaviours, and performance capabilities [Benyon, 1984; Chin, 1986; Quinn & Russell, 1986; Rich, 1983; Self, 1974; Sleeman, 1985; Witten, Greenberg, & Cleary, 1983]. The user information allows the interface to be aware of individual user needs and requirements. As a result, a system can better accommodate individual differences. *Adaptation* allows interfaces to adapt system behaviours to changing user needs and to reduce the amount of user effort that is expended. Therefore, a system can better accommodate variable user behaviours. *Facilitation* eases the cognitive and physical overhead associated with user interactions and ensures that user interactions proceed relatively smoothly. It is an important component of computer systems because it addresses concerns related to the support of complex and difficult user interactions.

History Tools : A Proposed User-Support Candidate

Computer-based user support is available, generally, in the form of help systems [Elkerton, 1988; Lee, 1992a]. This thesis proposes a different user support facility known as a *history tool*. A *history* (also known as user history) is a log of a user's past interactions. A *history tool* permits users to refer to their history and to incorporate parts of their history into the context of their current interactions. There are several reasons why history-based, user support tools are important and merit investigation:

- User support capabilities rely on history information.
- Humans draw heavily from past experiences.
- History-cued problem solving facilitates learning.
- History information helps users cope with improvisation.
- History tools relieve cognitive and physical burdens.

User Support Capabilities Rely on History Information

A history is a valuable source of information for the three basic user support capabilities (i.e., user models, adaptation, and facilitation). It contains descriptive, interpretive, and functional information. Descriptive information is fundamental to the operation of user models. It includes, for example, preferences, style of interaction, user skill level, and task knowledge. Interpretive information helps users to diagnose, explain, and understand problematic situations. It includes, for example, errors, feedback, and diagnostics. Functional information provides users with the means to adapt system behaviour and to facilitate user-computer interactions. It includes, for example, information about how users interact with a system, how they adapt to a system, and how a system facilitates the execution of user tasks. The interpretive and functional types of information are useful for adaptation and facilitation.

Systems have used history information as a crucial element of user support tools. However, these efforts have focused on identifying the functionality of history tools rather than characterizing the types of history information and the uses of history information. We argue that such characterizations are important as they reveal the value of and the range of uses of history information.

Humans Draw Heavily from Past Experiences

Humans draw heavily from their vast store of knowledge and experiences. This human characteristic is exploited by two different machine learning research efforts. In *case-based reasoning*, a reasoner uses memory of past cases within the same domain, sharing similarities to the current situation, to interpret or to solve a new case [Rissland, Kolodner, & Waltz, 1989]. *Reasoning by analogy*, like case-based reasoning, uses analogous situations, but not necessarily from the same domain, as the basis for reasoning about a problem.

In the user support context, a history contains information about past interactions. This information can be used to improve the quality and efficiency of user interactions. Users can avoid past errors, forego fruitless approaches attempted previously, or bootstrap onto a previous solution and thereby focus quickly on important aspects of a solution which requires user attention.

History-Cued Problem Solving Facilitates Learning

Studies of problem-solving mechanisms reveal that the cognitive processes required for problem solving and learning do not overlap sufficiently for learning to occur during problem-solving [Sweller, 1988]. In fact, Sweller (1988) suggests that a *means-ends* problem-solving strategy interferes with learning by preventing problem solvers from learning essential aspects of a problem's structure. A means-ends strategy involves working backward iteratively from a goal state by setting subgoal states until the problem state is reached. It is extremely effective for problems in which problem states can be specified as goals because it encourages a problem-solver to attend to differences between a current problem state and a goal state. However, such selective attention may totally ignore relationships between a problem state, its particular category of problem states, and

the particular operators. Thus, when problem solvers devote full attention to a goal, they may exclude those features of a problem necessary for learning.

Unlike a means-ends strategy, a *history-cued* strategy works with goal-specific and non-goal-specific problems (e.g., concept-discovery tasks). It uses actions from previous problem-solving episodes to generate actions for the current problem-solving episode. Sweller, Mawer, and Howe (1982) propose that a history-cued strategy makes use of rule induction, a primary means of knowledge transfer. Thus, a history-cued strategy could facilitate problem solving as well as learning.

A series of studies examined the issue of knowledge transfer using goal-specific problems solvable by either strategy: a rule-induction procedure (i.e., history-cued) or a non-rule-induction procedure (i.e., means-ends) [Sweller, Mawer, & Howe, 1982]. In situations where subjects chose a means-ends strategy, there was a comparative lack of transfer due to a failure to induce a rule. On the other hand, when subjects used a history-cued strategy, large performance changes on transfer tasks were observed. Insofar as this transfer indicated expertise or knowledge of a problem structure, use of a history-cued strategy resulted in more rapid acquisition of knowledge. Similar results were obtained in experiments where problem-solvers were presented with problems that had been modified to eliminate specific goals; problem-solving expertise was enhanced more rapidly (experiment referenced in [Sweller, 1988]).

These findings suggest that a history-based, user support tool may encourage the use of a history-cued problem-solving strategy rather than a means-ends strategy and thus, facilitate rule induction and knowledge transfer. Furthermore, such a tool may indirectly result in more rapid acquisition of knowledge.

History Information Helps Users Cope with Improvisation

A history contains knowledge and actions relevant to the particulars of the current situation (e.g., what actions were taken to arrive at the current situation). It also contains information about a user's evolving plan of action. However, unlike the plans that users have in their heads, user histories reflect improvisations made to their mental plans in order to fit the current situation; user histories are plans that are concrete, externalized, and directly accessible. Thus, they can help users figure out how they dealt with uncertainties in the past by relating the knowledge and actions needed during periods of improvisation. The relationship between a user history and plans-as-communication¹ and the ways in which a history can be enriched to support plans-as-communication are unexplored.

¹ Plans-as-communication relate the knowledge and actions needed during periods of improvisation (i.e., situated actions) [Agre & Chapman, 1990]. A user history is an example of plans-as-communication resource which helps users figure out how they dealt with uncertainties in the past.

History Tools Relieve Cognitive and Physical Burdens

User biases for the path of least cognitive resistance and repetition in user behaviour suggest strong user preferences for what has worked in the past, even if it is not an optimal approach. Users prefer to concentrate on conceptual difficulties associated with task execution rather than the tedious, mundane, and operational aspects of task execution. By reusing past approaches, users can rapidly repeat actions that are unchanged in the current task. Furthermore, users can benefit from improvements in their interactions as limited resources are more accessible and more effectively used.

Problem Statement

My thesis is that history, and tools based on history, can enhance user support. However, despite the potential benefits that both offer, very little attention has been focused on either and on their prospects for user support. In fact, there is little empirical evidence to illustrate the effectiveness of history tools and few studies and design efforts concerning history-based user support.

This dissertation investigates, through a series of studies, a number of specific questions related to the composition of a history and the prospects of history tools for user support. The dissertation explores deficiencies, potentials, and constraints of history and history tools. Each study poses specific questions concerning the effectiveness of history tools for user support. The studies examine the usefulness of current history tools, the extent of history-tool usage, the user interaction behaviours that can benefit from history tools, the ways that history tools can exploit such user behaviours, and the mental and physical costs associated with using history tools.

Dissertation Roadmap

The dissertation is divided into three parts. The first part, Chapters 2 and 3, characterizes the uses of current history tools and proposes requirements and options for enhancing the design of history tools. The second part, Chapters 4 and 5, describes an empirical observation and analysis of actual, everyday, natural user interactions with UNIX and UNIX history tools. The third part, Chapter 6, examines the effort involved in issuing repetitive user actions.

Chapter 2 focuses on how current history-like tools can enhance user interactions as well as the current state of this support. Various basic uses of a user history and their manifestations are characterized. System capabilities provided to support these uses are critiqued. Then, a number of existing systems are assessed in terms of the degree to which they support and integrate these uses.

Chapter 3 examines the needs and requirements for history tools which integrate the various uses of history. The kinds of history information and the kinds of functionalities that a history tool should support are proposed.

Chapter 4 is concerned with user interactions supported by history tools and user interactions with history tool usage. The findings from an exploratory study conducted with these concerns in mind are presented.

Chapter 5 examines in a formal way the user behaviour – user repetition of user actions – found in Chapter 4 and Greenberg and Witten (1988b)'s study. This behaviour is akin to behaviour observed in a program's references to computer memory. We adopt the concept that describes such program reference behaviours as a characterization of the repetition of user actions in our analysis of user interactions within a UNIX environment. Findings from a number of studies examining questions concerning this user behaviour and its implications for history-based, user support tools are presented.

Chapter 6 presents an analysis and comparison of the human information processes involved in issuing a recurrent command. The mental and physical effort involved in typing the recurrent command is compared to the mental and physical effort involved in using history tools. The comparative measures are the number and variety of different human information processing operations involved in using each proposed design. Since our comparative analysis relies on the predictions generated by cognitive models, it is important that these predictions are accurate. We conclude by examining the accuracy of these predictions.

Chapter 7 summarizes the dissertation and our research contributions. A number of future research directions for history as a user support tool are discussed.

Chapter 2

Uses of Information in a History

This chapter presents seven basic uses of the information in a user's history which can potentially enhance user-computer interactions. These uses are based on a survey of a number of history-like tools available in current systems. A taxonomy is used to structure the presentation of these uses, their various manifestations, and their system support. In addition, the state of current history tools is examined with respect to 1) user needs for each manifestation of history use and on system capabilities provided in support of each and 2) support of these uses of history in a number of general-purpose development systems.

Before proceeding, we need to clarify our use of several terms in this dissertation. The terms *history* and *user history* are used interchangeably to refer to a collection of information recorded from an earlier part of a user's interactions. The term *script* refers to a sequence of user actions to be carried out [Archer, Conway, & Schneider, 1984]. If in the course of script recording, user actions are actually performed, then the recording (aside from being a script) is part of a history. A future instantiation of a script is a *future part of a history*. Viewing a user's interactions with respect to an activity timeline, actions and objects that took part earlier are candidates for a history and actions and objects that are to be performed in the future are candidates for a future part of a history.

Taxonomy

There have been two previous surveys of history tools. Greenberg (1988b) examined different interaction styles in history tools for facilitating the reuse of computer tools. Linxi and Habermann (1986) examined different uses of the information in a history to facilitate the software development process. Like Greenberg (1988b), we are interested in how history tools facilitate user interactions but like Linxi and Habermann (1986), our survey enumerates different uses of history information. Our survey results include uses of history information that are not evident in the software development task domain. Furthermore, we partition our survey results into a taxonomy of *uses of history*. Each *use of history* is an abstraction of a designer's conception of a user's *usage intention* (e.g., reuse or error recovery).

Our taxonomy characterizes each *use of history* in terms of two variables: *type* and *system support*. Examples of systems are provided for the various system support associated with a particular type of history use (see Table 1).

Uses	Type	System Support	Example Systems
reuse	a script literally		CONMAN [Haeberli, 1986] PLAYERPIANO [Bier & Freedman, 1985]
	a previous interaction with possible modifications	list of accepted operations descriptive manipulation	C Shell [Joy, 1980], INTERLISP-D USE, REDO [Teitelman & Masinter, 1981]
		direct manipulation	KORN Shell [Korn, 1983], HCR Hi [HCR Corporation, 1987], MINIT [Barnes & Bovey, 1986], TC Shell [Ellis, Greer, Placeway, & Zachariassen, 1987], INTERLISP-D FIX [Teitelman & Masinter, 1981]
		display's contents	CEDAR [Teitelman, 1985], EMACS [Stallman, 1981], SMALLTALK-80 [Goldberg, 1984], SUNTOOLS [SUN Microsystems, Inc., 1986]
operations as a functional group	macro history macro recorded macro		ALOE [Linxi & Habermann, 1986] EMACS [Stallman, 1981], HP NEWWAVE AGENT [Stearns, 1989], MACROS BY EXAMPLE [Olsen & Dance, 1988], MEVEC [Ash, 1981a], TEMPO [Whitby, 1986], QUICKEYS [Bobker, 1988]
		programming with example by example	SMALLSTAR [Halbert, 1984] METAMOUSE [Maulsby, Witten, & Kittlitz, 1989] PERIDOT [Myers & Buxton, 1986]
inter-referential I/O	relate previous I/O to current input		SYMBOLICS [McMahon, 1987]
error recovery	system recovery	restore system state undo erroneous operations	SMALLTALK-80 [Goldberg, 1984] US&R [Vitter, 1984], EMACS [Stallman, 1981], CHIMERA [Kurlander & Feiner, 1988]
	command correction	descriptive manipulation direct manipulation	C Shell [Joy, 1980] INTERLISP-D HISTMENU & FIX [Teitelman & Masinter, 1981], KORN Shell [Korn, 1983], TC Shell [Ellis, Greer, Placeway, & Zachariassen, 1987]
navigation	where am I, where did I come from, where have I been	information spaces	HYPERCARD RECENT [Goodman, 1987], TIMELINE PAGE [Feiner, Nagy, & van Dam, 1982], WHAT, WHERE, WHENCE [Engel, Andriessen, & Schmitz, 1983]
		activity spaces	SITES, MODES, and TRAILS [Nievergelt & Weydert, 1980], Room Stack in ROOM MODEL [Chan, 1984], Back Door in ROOMS [Card & Henderson, 1987]
reminding	user consultation		SUNTOOLS [SUN Microsystems, Inc., 1986], JOBS [Joy, 1980], SITES, MODES, and TRAILS [Nievergelt & Weydert, 1980]
	visual cues	display user history	INTERLISP-D HISTMENU [Teitelman & Masinter, 1981], SUNTOOLS [SUN Microsystems, Inc., 1986], HYPERCARD RECENT [Goodman, 1987], TIMELINE PAGE [Feiner, Nagy, & van Dam, 1982]
user modelling			UKNOW [Desmarais & Pavel, 1987], UNIX CONSULTANT [Chin, 1986], STEREOTYPE [Rich, 1983]
user interface adaptation		default menu selection predict next action	SUNTOOLS [SUN Microsystems, Inc., 1986] REACTIVE KEYBOARD [Witten, Cleary, & Darragh, 1983]

Table 1: Taxonomy of uses of a history along with example systems.

The *type* variable characterizes different manifestations of a usage intention. To illustrate, reuse may be manifested as the literal reuse of a group of user actions – *reuse a script literally* – or as the reuse of an object/action with some modifications – *reuse a previous interaction with possible modifications*. Each manifestation suggests a set of user needs. For example, users who want to *reuse operations as a functional group* need to group several operations together for invocation as a unit.

The *system support* variable characterizes system capabilities that are provided to meet certain user needs of a particular type of history use. System support varies in terms of the sophistication of the user support provided and reflects the amount of user effort required to realize the particular type of use of history. For example, some designers provide a *macro* facility to support *reuse operations as a functional group*, while other designers provide an *example programming* facility. A macro allows users to associate a sequence of actions to a single action while an example programming facility allows users to generalize an example execution trace to a program.

Our taxonomy is the result of a survey of current history-like tools that directly enhance a user's interactions with a system. While one may envision different types of uses of history to support user interactions, the taxonomy only considers those supported by existing systems and those uses of history that fall inside the scope of interest. For example, user traces for analyzing the design of a system are not considered a direct user support capability.

The taxonomy characterizes the value of the information maintained in a history in terms of seven basic usage intentions. The taxonomy provides a framework for presenting and critiquing user needs and system support for a particular type of use of history. Furthermore, the usage intentions and their various manifestations and implementations provide a context for examining how well current general-purpose systems support these seven uses of history. Finally, the taxonomy provides a structure for thinking about the kinds of information and functions required to use such history information. The next chapter presents the insights into design requirements for multi-use history tools extracted from the taxonomy.

History for Reuse

Currently, the most common use of a history tool is to reuse and possibly modify a history item to save keystrokes and/or mouse strokes [Linxi & Habermann, 1986]. Different types of reuse are supported in current history tools.

Reuse a Script Literally

An early form of reuse originates from early scripting systems where the same sequence of actions – recorded during the performance of an earlier task – is replayed literally in the current context. The recording “pushes the buttons” as if the user is there doing it – similar to a player piano. Since it is intended for situations in which the same operations are to be performed each and every time, this type of reuse is both primitive and limited.

Reuse an Interaction with Possible Modifications

Earlier actions and objects may also be reused after some modifications. System support varies depending on the contents of a history: *list of accepted operations* or *display's contents*.

A list of *accepted* operations (i.e., operations that were successfully executed), is a user history kept by systems that model textual interactions as *typescripts* (transcript of input and output). History items are selected and modified using one of two interaction styles: *descriptive manipulation* or *direct manipulation*. A descriptive manipulation style requires users to remember history items and history manipulation syntax and to combine a description of the history item to be retrieved and the changes to be made to it into a single request (e.g., in C Shell `!-5:s/aa/bb` recalls the 5th last command substituting the first occurrence of *aa* by *bb* [Joy, 1980]). In a direct manipulation style, a user history is displayed to remind users of its contents and to allow them to point to the desired item (see Figure 1 for a sample display of the INTERLISP-D HISTMENU). Furthermore, history items are first retrieved before they can be edited, thus simplifying select and modify operations.

In systems where a user can copy and edit text appearing anywhere on the display, the history is the *display's contents* – accepted and unaccepted user inputs and system responses (see Figure 2). The system uses the display for mediating user-computer interactions, as well as, supporting history manipulation

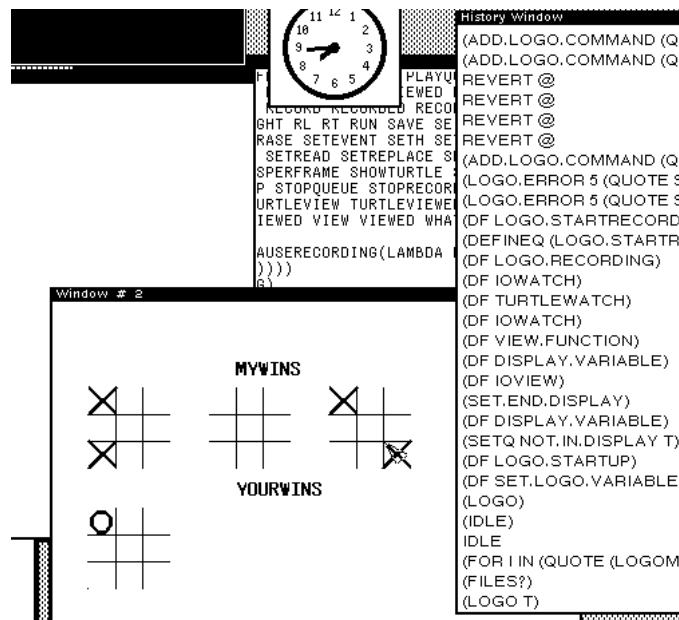


Figure 1: INTERLISP-D's HISTMENU displays a history of the commands issued to the Executive in the form of a menu. The user may select the items from the menu (the window entitled *History Window*).

```

shelltool - /bin/newcsh
104 OIS.DB% cd ..
105 OIS.DB% date
Fri Apr 21 17:57:36 EDT 1989
106 OIS.DB% Make history qms db big draft
Make: Command not found.
107 OIS.DB% pwd
/homes/ois/wind/alee/thesis
108 OIS.DB% cd proposal
109 OIS.DB% ls -l history
-rw-r--r-- 1 alee      58397 Apr 20 14:35 history
110 OIS.DB% █

```

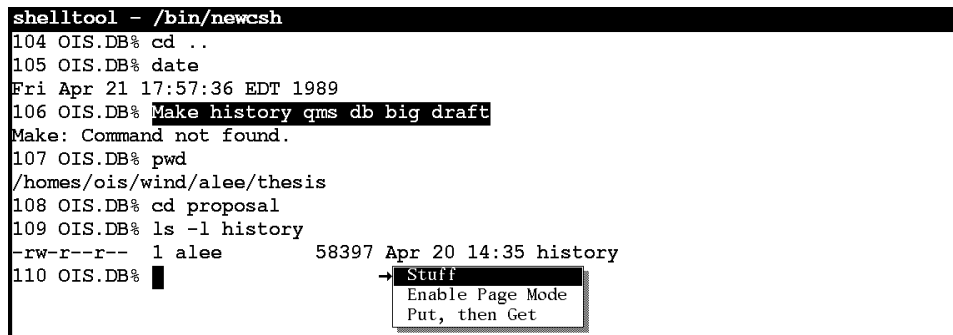


Figure 2: In SUNTOOLS, text may be selected, copied, and stuffed to the input focus (i.e., *cursor*).

capabilities (e.g., display, selection, and modification) [Scofield, 1981; Young, Taylor, & Troup, 1988]. Browsing, rather than querying, is the means of examining a history. Such history manipulations are manual, but direct. Furthermore, a broader notion of the constituents of a user history is supported by this approach as compared to the typescript approach. More importantly, since the display is an integral part of the interaction and browsing and editing are basic system functionalities, no additional history machinery is required to support this particular type of history use; it is available virtually for free.

Reuse Operations as a Functional Group

Users may reuse a set of operations as a compound command. Two kinds of system support are available: *macro* and *example programming facilities*.

A macro facility permits a user to expand one instruction, command, keystroke, or mouse action into a sequence of user actions [Ash, 1981a]. Current macro facilities in graphical systems do not usually permit users to view and edit a macro. Two kinds of macros can be created using parts of a user history: *history macro* and *recorded macro* [Linxi & Habermann, 1986]. A *history macro* is constructed by selecting previous operations from a user history while a *recorded macro* is constructed by first entering a record mode, issuing the operations, and then stopping the recording. Figure 3 illustrates an example construction of a recorded macro in TEMPO [Whitby, 1986]. Unlike a history macro facility, where users can create macros after issuing the relevant user operations, users creating recorded macros must know in advance of issuing the relevant user operations that they want to create a macro.

An example programming facility allows the user to construct a program from a recording of the user actions used to perform an example task. The program is a generalization of the example task history. Generalizations are made either by the user – *programming with example* – or by the system – *programming by example* [Myers, 1986]. A *programming with example* system provides the user with programming and editing tools to generalize an example task history. Such augmentations require a knowledge of, and competence in, programming. Figure 4 illustrates an example construction of a program in SMALLSTAR [Halbert, 1984].

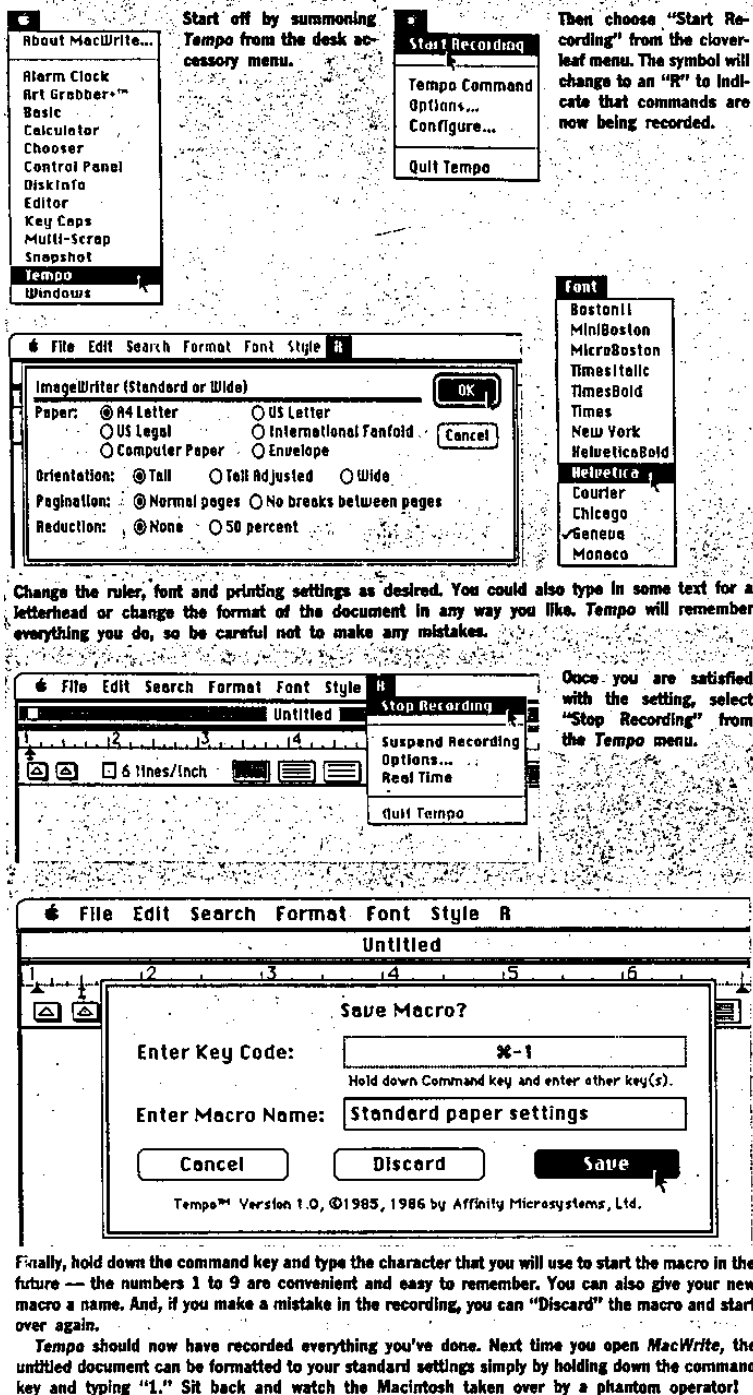


Figure 3: An example showing the construction of a TEMPO macro to automatically format a blank MACWRITE document to standard settings [Whitby, 1986].

In the mail form, the **Weight** field is examined to determine what should be filled in the **Class** field.

The initial program after the user records the actions assuming that the value in the **Weight** field is less than 1 pound. The user selects the statements that will be in the body of the conditional to be provided.

This shows the completed program with the correct conditional test that the user has explicitly added.

Figure 4: This SMALLSTAR program determines whether a customer's order is to be sent by first-class or fourth-class mail [Halbert, 1984]. If the value in the **Weight** field of the mail is less than one pound, the order will be sent by first-class mail, otherwise by fourth-class mail.

In a *programming by example* system, the system makes the procedural inferences and generalizations based on observations of user actions for a number of related example tasks and knowledge about the specific domain (e.g., METAMOUSE deduces graphical procedures from user execution traces in a drawing program [Maulsby, Witten, & Kittlitz, 1989]).

History for Inter-referential I/O

In human dialogue, conversants make abbreviated references to objects and actions that took place earlier in the dialogue [Draper, 1986; Reichman-Adar, 1986]. For example, two conversants have just attended a talk given by speaker A and in their subsequent conversation about the talk, they use the referent, *the talk*, to refer to the talk instead of using an elaborated reference. Similarly, two persons (A and B) are examining the contents of a file directory on a computer system and person A decides to relocate one of the files to another file directory and then person B tells person A to *do the same* for another file; a request which person A can interpret.

Inter-referential I/O is the computing analogue of this conversational capability [Draper, 1986]. It allows the user's input or the system's output to reference previous input and/or output available on the display. Current systems support part of this capability. Specifically, users can make explicit the links between actions and objects from an earlier part of the dialogue with those from the current part of the dialogue (i.e., *relate previous input/output to current input*). This enables users to make abbreviated associations which the system can disambiguate and interpret. However, users are unable to relate previous input/output to current output.

This use of history involves more than the simple reuse of parts of a user history, it allows a user to use abbreviated referents to relate pieces of past and present interactions. Such tools are able to track and interpret these referents because they share a common interaction context with the user much like the shared context between two human conversants. Hence, user references can be explicitly abbreviated and systems are able to disambiguate them.

The SYMBOLICS GENERA programming environment [McMahon, 1987] highlights display objects that are suitable as inputs to the current user command as the user moves the mouse over them (see Figure 5). This is made possible by a type mechanism which relates application objects, including those on the display, with the way they are to be used in particular user interface situations. Thus, the system has a structured context and an abbreviated referencing scheme for linking objects required in a user action with those on the display much like the shared context and abbreviated referents available in human conversations.

History for Error Recovery

Error recovery tools allow users to recover from errors (e.g., typing). Two different types of error recovery are intended and supported: *command correction* and *system recovery*. Command correction is simpler than system recovery.

```

:Show Directory (files [default Q:>Zippy>>.s.newest]) Q:>Zippy>>.s.newest
Q:>Zippy>>.s.newest
13161 free, 541529/554698 used (97%, 7 partitions) (LMFS records, 1 = 4544. 8-bit bytes)
a.lisp.1      1      7(8)      05/18/87 18:15:55 (05/18/87)
b.lisp.1      1      9(8)      05/18/87 18:16:06 (05/18/87)
c.lisp.3      1     19(8)      05/18/87 18:48:28 (05/18/87)
ed.lisp.1     1    238(8)     04/30/86 12:38:38 (04/30/86)
pins.directory.1 1  DIRECTORY | 06/18/87 17:52:58 M=08/18/87
proprietary-information.directory.1 1  DIRECTORY | 06/18/87 18:00:12 N=08/18/87 RH
test.bin.1   1    377(16)   01/16/86 22:26:38 (01/16/86)
test.lisp.1  1    151(8)    01/16/86 22:26:34 (01/16/86)
test.umbin.1 3  10093(8)  01/16/86 22:27:01 (01/16/86)
zmail-init.lisp.1 1  639(8)   12/30/85 18:43:32 (08/18/87)

12 blocks in 18 files

:Hardcopy File (file [default Q:>Zippy>>zmail.text.newest]) █

```

Figure 5: A user of the SYMBOLICS GENERA programming environment [McMahon, 1987] lists the contents of the current directory and then wishes to print out one of the files in this directory, using the Hardcopy command. The objects on the display with a type (i.e., filename) corresponding to the desired operand type for the Hardcopy command are highlighted for selection as the user moves the cursor over them.

Command Correction

Occasionally, a user's directive is rejected by the system because the directive is incorrectly typed, incorrectly constructed, or inappropriately issued. In such cases, command correction in the form of a FIX and REDO facility allows users to correct the error before the directive is re-issued. In a *descriptive manipulation system*, the facility provides a syntax for describing the modifications to be made to the recalled history event (e.g., in C Shell, !!s/aa/bb substitutes the first occurrence of aa with bb in the last command line issued). In a *direct manipulation system*, the user copies the incorrect command directive and edits it before issuing it.

System Recovery

A system recovery tool allows users to recover from an erroneous change to the system state. System recovery may be performed in one of two ways: *restore system state* or *undo erroneous operations*.

System recovery may be performed by restoring the system state to a snapshot of an earlier system state. SMALLTALK, which maintains snapshots as well as records of changes made since the last snapshots, allows users to backtrack to an earlier system state and incrementally re-execute the changes [Goldberg, 1984]. The frequency and automaticity of making snapshots affect the ease with which a user can restore a system state. Furthermore, a recovery process that allows users to roll a restored system state forward to the point before the error, using the record of the changes made between snapshots, may involve a lot of manual changes, especially if snapshots are made infrequently.

Alternatively, users may undo an erroneous operation. Three different variations of such a facility exist [Vitter, 1984]. The most sophisticated form, known as *tree UNDO/REDO/SKIP*, supports the following error recovery operations:

- Remove erroneous commands
- Insert missing commands
- Substitute for erroneous commands
- Re-arrange the order of commands
- Change of heart (UNDO an UNDO)

A history contains the primitive user commands that have been issued and is maintained as a tree-like data structure with a pointer variable. The commands, UNDO, REDO, and SKIP, allow users to undo, redo, and skip over previously issued commands by reorganizing the links in the tree-like history data structure. Figure 6 illustrates two sets of error recovery scenarios using these commands.

History for Navigation

In the course of using a computer system, users commonly ask questions like where they are, where they just came from, and where they have been [Engel, Andriessen, & Schmitz, 1983; Fitter, 1979; Nievergelt & Weydert, 1980]. The information in a history helps users to answer these questions. Depending on whether the user is navigating through an *information space* or *activity space*, a system provides different navigation aids.

In navigating through large information spaces, users can become lost [Mantei, 1982]. Thus, such systems typically maintain system snapshots corresponding to places a user visited, generally in chronological order (see Figure 7).

An activity space contains all user activities for a particular task. Users are known to juggle several tasks by moving back and forth between activity spaces. Occasionally, they lose track of their task switching [Bannon, Cypher, Greenspan, & Monty, 1983; Card & Henderson, 1987; Cypher, 1986; Miyata & Norman, 1986]. ROOMS' BACK DOOR [Card & Henderson, 1987] allows users to find out about the last activity space they worked in while ROOM MODEL's ROOM STACK [Chan, 1984] lists the activity spaces the user visited in chronological order.

History for Reminding

The contents of a history may remind users of past knowledge and events. A reminding is deliberate or spontaneous activation of past knowledge and experiences and integration of that information into the current context [Ross, 1989]. A deliberate reminding is supported by *user consultation* of a history while a spontaneous reminding is supported by *visual cues* in a history. The effectiveness with which history tools support these two types of reminding depend largely on whether the tools provide good visual cues as well as browsing and querying tools.

User Consultation

A deliberate reminding occurs when users query or consult their history to find specific information. For example, users can examine their history:

$$(A) \quad A_1 \leftrightarrow A_2 \leftrightarrow \dots \leftrightarrow A_{n-1} \leftrightarrow A_n$$

last_done \nearrow

The user realizes after issuing commands A_1, A_2, \dots, A_n that commands B_1 and B_2 must be inserted between A_1 and A_2 .

$$(B) \quad A_1 \leftarrow \cdot \cdot \rightarrow A_2 \leftarrow \cdot \cdot \rightarrow \dots \leftarrow \cdot \cdot \rightarrow A_{n-1} \leftarrow \cdot \cdot \rightarrow A_n$$

\uparrow
last_done

Commands A_n, A_{n-1}, \dots, A_2 are undone (signified by dotted arrows).

$$(C) \quad A_1 \leftarrow \cdot \cdot \rightarrow A_2 \leftarrow \cdot \cdot \rightarrow \dots \leftarrow \cdot \cdot \rightarrow A_{n-1} \leftarrow \cdot \cdot \rightarrow A_n$$

\uparrow
 $\rightarrow B_1 \leftrightarrow B_2 \leftarrow$ *last_done*

Commands B_1 and B_2 are issued. They appear as new nodes on a new branch emanating from the node A_1 with solid arrows.

$$(D) \quad A_1 \leftarrow \cdot \cdot \rightarrow A_2 \leftrightarrow \dots \leftrightarrow A_{n-1} \leftrightarrow A_n$$

\uparrow
 $\rightarrow B_1 \leftrightarrow B_2$ *last_done* \nearrow

Commands A_2, \dots, A_n are reinstated with n-1 REDOs. At this point, the user realizes that the original sequence in step (A) is the correct sequence.

$$(E) \quad A_1 \leftarrow \cdot \cdot \rightarrow A_2 \leftarrow \cdot \cdot \rightarrow \dots \leftarrow \cdot \cdot \rightarrow A_{n-1} \leftarrow \cdot \cdot \rightarrow A_n$$

\uparrow
 $\rightarrow \cdot \cdot \rightarrow B_1 \leftarrow \cdot \cdot \rightarrow B_2$
last_done

The user undoes commands following A_1 .

$$(F) \quad A_1 \leftrightarrow A_2 \leftrightarrow \dots \leftrightarrow A_{n-1} \leftrightarrow A_n$$

\uparrow
 $\rightarrow \cdot \cdot \rightarrow B_1 \leftarrow \cdot \cdot \rightarrow B_2$ *last_done* \nearrow

The user invokes REDO to redo A_2 and then issues n-2 REDOs to redo commands following A_2 .

Figure 6: (A) illustrates the state of the tree-like history data structure before the user performs a sequence of changes (B) to (F) related to two error recovery operations. (B) to (D) illustrates insertions of two missing commands. (E) to (F) illustrates a change of heart; the user wants to undo changes made in steps (B) to (D). Note, commands with dotted arrows are undone while commands with solid arrows are issued.

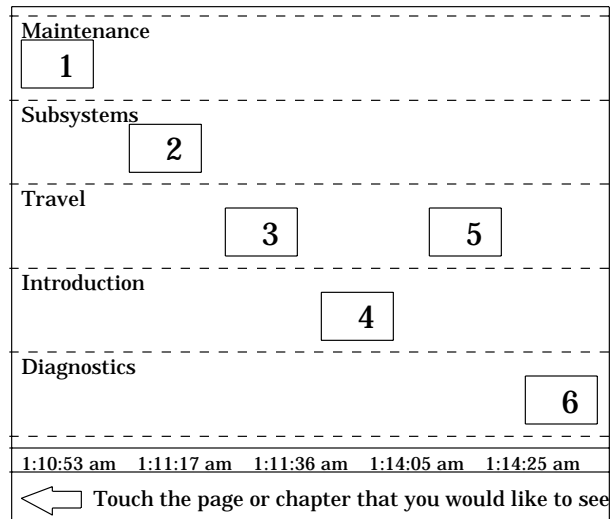


Figure 7: In Feiner, Nagy, and van Dam (1982)'s system, the last 6 examined pages are displayed chronologically as miniatures along their parent chapter's band. Selecting the miniature returns the user to that page. Other pages in the timeline are displayed by selecting the scroll arrow.

- to identify activities requiring attention or causing errors,
- to find relevant information from a previously related task, or
- to re-acquire the mental context for an interrupted activity.

An important design consideration is the user effort required to cull the information from a history and the extent of the system support available to assist such user efforts (e.g., query and browse). Without appropriate system support, this type of history for reminding is limited and users must expend considerable effort to extract the desired information.

Visual Cues

The display of a user history provides visual cues that trigger the spontaneous recall of relevant past information and activities that the user would otherwise not remember. More importantly, reminders of past activities may lead a user to consult the history in more detail; an activity which would never have been pursued without the benefit of the reminding. While a history provides the information, visual cues are crucial to the spontaneous activation of a reminding. Thus, it is important to provide appropriate support for such visual cues and to develop innovative and effective visual representation schemes. Current history-tool support for this history use consists largely of displaying portions of the user history with little or no novel visual cues.

History for User Modelling

A user history may contain information about a user's skill level, task knowledge, preferences, and personality traits. When systems infer such information from a history to formulate user models, they are using *history for user modelling* [Chin, 1986; Desmarais & Pavel, 1987; Rich, 1983; Tyler & Treu, 1986]. The user model may then be used to adapt system behaviour or to provide user-tailored assistance. Figure 8 illustrates the kinds of information that UKNOW infers from a user history [Desmarais & Pavel, 1987]. Systems use various techniques to extract user information including deterministic and probabilistic approaches, behaviour to structure transformations, and induction and knowledge inferences. Current history tools for user modelling are limited by our understanding of user characteristics that differentiate individuals and by the techniques for inferring user characteristics.

History for User Interface Adaptation

Using a user history and appropriate heuristics, a system can adapt system behaviour to suit user needs, to predict user actions, or to infer user preferences [Greenberg & Witten, 1988b]. A simple example is a menu system whose default selection is the last menu selection. Another example is the REACTIVE KEYBOARD which predicts what the user is going to type based on an adaptive model of the text that has been typed previously [Witten, Cleary, & Darragh, 1983].

Like programming-by-example systems, user-interface-adaptation tools are successful within specific application environments because they can exploit application knowledge and use knowledge-based techniques to interpret user behaviours. However, it is difficult to exploit application knowledge in a general-purpose environment. Therefore, general-purpose history tools for user interface adaptation have not fared as well. A more viable approach is to use knowledge-based or probabilistic techniques to infer behavioural semantics from a user history (e.g., use rules to interpret observed behaviour [Quinn & Russell, 1986]).

Assessment

While the preceding section presented user needs and critiqued the level of system support available for different types of uses of history, this section examines the uses of history that are directly supported by current systems. To facilitate this examination, a number of interactive systems are selected and assessed in terms of how well they directly support all seven uses of history.

In order to provide a fair assessment of the state of current support for all uses of history, only general-purpose development environments are considered rather than special-purpose systems (e.g., text editors, information retrieval systems). Such environments support access to different applications (e.g., text editing, graphical drawing, electronic communication, programming, and managing file systems). We picked six systems that supported some form of a history tool for our assessment: ALOE, INTERLISP-D, MINIT, MPW, ROOMS, and SEED. Appendix A provides a brief description of these systems.

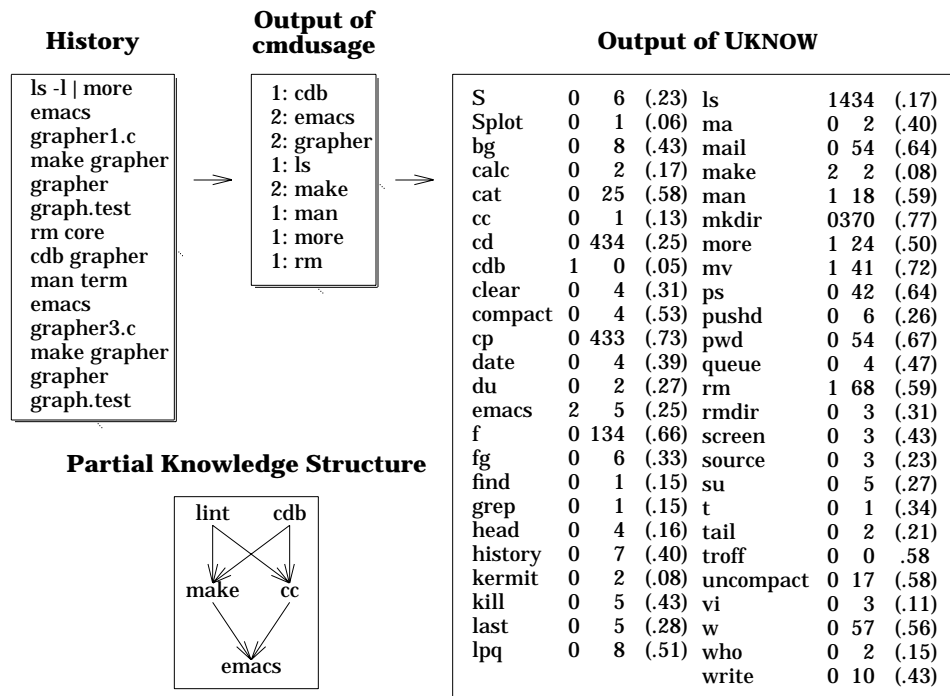


Figure 8: UKNOW uses three techniques to deduce commands known by a user: observation, inference, conditional probability [Desmarais & Pavel, 1987]. The first two techniques are based on analyses of a user history. One metric, appearing in the first column of numbers in the UKNOW output and in the “cmdusage” output, is the observed frequency of use of each command. A second metric, appearing in the second column of numbers in the UKNOW output and derived with the aid of a knowledge structure, is the number of inferences about a user’s mastery of a given command based on a user’s mastery of other commands. A knowledge structure links sets of UNIX commands together with implication relationships. In the partial knowledge structure, observation of *lint* will result in one inference of mastery to both *make* and *cc* and in two inferences to *emacs*. The third metric is the conditional probability that a command is known by users given that their mastery of the command is neither observed nor inferred (note that the first two metrics are 0). Mastery of *troff* was determined using this conditional-probability measurement technique (i.e., third metric is not enclosed by parentheses to indicate that its value is derived from the conditional probability).

Results and Discussion

Each system is assessed in terms of whether it *does* or *does not* directly support each type of use of history. Table 2 summarizes the results of the assessment. At a glance, no single system supports all seven uses of history. These systems support some manifestation of reuse, error recovery, navigation and reminding.

Uses of History	ALOE	INTERLISP-D	MINIT	MPW	ROOMS	SEED
reuse						
operations literally	●	○	○	○	○	●
a previous interaction with possible modifications	●	●	●	●	●	●
operations as a functional group	●	○	○	○	○	○
inter-referential I/O	○	○	○	○	○	○
error recovery						
command correction	●	●	●	●	●	●
system recovery	●	●	○	○	●	●
navigation						
where am I	○	○	○	●	●	○
where did I come from	○	○	○	○	●	○
where have I been	○	○	○	○	○	○
user reminders						
user consultations	●	●	●	●	●	●
visual cues	○	○	○	○	○	○
user modelling	○	○	○	○	○	○
adaptive user interfacing	○	○	○	○	○	○

Table 2: Assessment of the uses of history information directly supported by eight general-purpose environments. A ● means the *system does support* the use while ○ means the *system does not support* the use.

History for user modelling, user interface adaptation, and inter-referential I/O are not supported by any of the six systems. In order to support such uses of history, a system must maintain application-specific information. Typically, such information is not maintained by a general-purpose development environment.

History tools also need to support the various types of history uses; otherwise, users may have to do a lot of fiddling to mimic these uses. For example, users who want to *reuse operations as a functional unit* but are provided with a tool to *reuse a previous interaction with possible modifications* must deal with each operation in the set individually rather than the set as a whole. All six systems support reuse of previous interactions with possible modifications, command correction, and deliberate reminding. Reminding is facilitated by history tools when these tools allow users to consult their history visually. ALOE and SEED are the only systems to support more than one type of history for reuse while MPW and ROOMS are the only systems to support some type of history for navigation. All systems, except MINIT and MPW, support both types of history for error recovery.

Current general-purpose history tools fail to support all seven basic history uses as well as all types of history uses within the seven basic categories. This failure is due to two factors: the kind of information maintained in a history and the history capabilities provided. An in-depth discussion of both factors is deferred until the next chapter.

Concluding Remarks

This chapter presents three important observations about the current state of history-like tools revealed by a survey of such tools. First, there are seven different basic reasons for using a history:

- (1) *History for reuse* – Users repeat operations to avoid re-entering operands and operations from scratch.
- (2) *History for inter-referential I/O* – Users make abbreviated references to objects and actions that took place in an earlier part of the dialogue.
- (3) *History for error recovery* – Users edit an incorrect action or undo the effects of an erroneous operation on a system.
- (4) *History for navigation* – Users consult a history to find out where they have been, where they have just come from, and where they are.
- (5) *History for reminding* – Users are reminded of relevant information associated with past activities.
- (6) *History for user modelling* – Systems infer or derive information about users by observing their activities in the course of their interactions.
- (7) *History for user interface adaptation* – Systems use history as a source of information to adapt automatically the interface behaviour to the user's needs.

Second, the taxonomy variables, *type* and *system support*, are used to characterize a use of history. The *type* variable characterizes the various manifestations of a basic usage intention. The *system support* variable characterizes capabilities that system designers have provided to address particular user needs. Both variables are useful, respectively, for thinking about the user needs posed by a particular type of history use and for thinking about the degree of system sophistication that is needed to support a particular type of history use.

Third, the taxonomy also provides a global context for assessing the extent to which current systems support the seven uses of history listed in Table 1. The assessment of six general-purpose development systems reveals that current general-purpose history tools fail to support all seven uses of history. Thus, users must either perform the desired operations manually or expend considerable effort to make the desired use of history fit the supported use of history.

This chapter introduces concepts that are relevant to the discussion in the next chapter concerning issues related to the design of history tools. Thus, the taxonomy provides a vehicle for not only critiquing current history tool designs but also thinking about the kinds of information a history tool should maintain and the capabilities a history tool should support.



Chapter 3

Cognitive and Behavioural Issues

According to Carroll, Kellogg, and Rosson (1991), current design practices in human-computer interaction involve a “cycle of discovering, defining, and understanding tasks people want to do, need to do, or might do, and then designing, developing, and analyzing artifacts to support tasks” (pg. 98). Such development activities are circumscribed by a task-artifact cycle which refines, emulates, and synthesizes HCI artifacts based on characterizations, analyses, and comprehension of tasks and artifacts during use. They suggest that the task-artifact cycle is a manifest structure of HCI technology evolution in which tasks and artifacts co-evolve: “a task implicitly sets requirements for the development of artifacts to support it; an artifact suggests possibilities and introduces constraints that often radically redefine the task for which the artifact was originally developed” (pg. 79).

The preceding and current analyses are part of an iteration through such a task-artifact cycle. Based on an examination of history-tool artifacts, the preceding chapter provides a rational reconstruction of seven independent tasks which early designers envisioned history tools supporting. It reveals that current history tools focus on providing support for individual uses of history information while paying little or no attention to the integration of various uses of history tools.

This chapter is concerned with the integration of the seven uses of history information to evolve new history-tool designs. The analysis focuses on cognitive and behavioural considerations for such integrated history tools. The first section proposes different kinds of information that a history should maintain. The second section proposes system capabilities needed for using this information.

Contents of a User History

The discussion, thus far, has been rather vague about the kinds of information maintained in a user history except to say that it is a collection of information associated with a user’s past interactions with a system. This was done deliberately to allow an unrestricted interpretation of a history use to dictate the kinds of information that a user history should maintain rather than to allow a haphazard choice of whatever is easily accessible.

The seven history uses provide insights into a variety of information that history tools should maintain. While certain history information may serve a number

of different purposes, it is unlikely that the particular information can service all seven history uses. Similarly, information available in one form may not be as useful as the same information represented in a different form. Therefore, it would be useful to maintain a variety of information related to a user's past interactions as well as a number of different representations of the same information. In the following subsections, we present a number of distinctions regarding different representations and different kinds of history information:

- Actions and objects referenced in actions
- User inputs and system responses
- States as well as transitions
- Specific information and peripheral information
- Externalized information and internalized information
- User and behavioural information

Actions and Objects Referenced in Actions

In addition to reusing previous actions, users may reuse the objects referenced in their previous actions (e.g., the name of a directory). Many history tools in UNIX maintain a log of user actions only (i.e., whole command lines) and do not keep separate logs of action objects (e.g., file names, directories, network addresses). Such information would facilitate search and selection. For example, the SYMBOLICS GENERA programming environment [McMahon, 1987] supports a type mechanism that maintains multiple lists of different objects using a *user interface type* and allows users to access easily a relevant list, where relevancy is determined by the desired type in the current context.

User Inputs and System Responses

History for inter-referential I/O also reveals that users, in addition to referencing previous user inputs, may also reference system responses produced from previous user actions. Maintaining a history of system responses is useful because users can consult or make use of the system responses in their subsequent

```
98 OIS.DB% ls /usr
  bin/      dict/      hosts/     local@    new/      sccs/     stand/
  crash/    etc/       include/   man/      pub/      skel/     tmp/
  Diag@    games@    lib/       msgs/     preserve@ spool/    ucb/

99 OIS.DB% !!skel/
```

Example 1: The 98th command reveals the contents of a directory, */usr*, which represents information generated as a result of a user request. One piece of the information, *skel/*, is used as part of a subsequent user command which requests the system to list the contents of the subdirectory *skel/*.

actions (see Example 1) without first having to repeat the user actions to derive the desired system response (i.e., history tools can economize user actions).

States As Well As Transitions

A distinction arising from our taxonomy is that a history can be a collection of visual snapshots of system states or a collection of state transitions (i.e., user operations used to effect state transitions). The visual snapshots contain information about particular system states. Such information is useful for helping users to re-acquire the mental context for a desired system state. However, a state-based user history does not maintain explicit information about the operations that led to a state from a previous state. Thus, it is not a suitable representation when users need access to the transition information.

Of course, there are also arguments to be made about the unsuitability of transition-based user histories. To illustrate, users need to examine some aspect of an earlier system state and they only have a transition-based history available to them. They must first have a mental image of either the current state or some earlier state. Then, using the transition-based history they must incrementally derive the previous or subsequent system state, depending on which mental image they start with, by updating that mental image until they reach the desired system state [Cowan & Wein, 1990]. Hence, a transition-based user history requires users to expend substantial mental processing effort in order to locate a system state so that they can examine some aspect of that system state. Thus, for this type of a task, the transition-based representation is not suitable when they need to access state information.

Specific Information and Peripheral Information

So far, we have concentrated on information *specific* to a user's interactions (e.g., user actions, system responses). However, history for reuse and history for error correction have also shed light on the need to maintain *peripheral information*. This is information, generated in the course of user interactions, that is not directly relevant but peripherally relevant to a user's immediate interactions. This information is important because it provides users with "information for thought" (i.e., information that can enhance both user interactions with a system as well as the user's understanding of their tasks and interactions). There are three types of peripheral information that should be maintained in a user history: *history commands*, *unaccepted user inputs*, and *snapshots*.

History commands are commands used to manipulate a history. They are considered peripheral to the immediate user interactions because they are not commands related to a user task but commands that expedite the specification of actions to perform a task (i.e., they allow users to reuse previous actions). No history tool maintains history commands, but they should because it may be advantageous for users to manipulate a history command rather than a task command (i.e., it may be faster and require less effort). Consider the scenario in Example 2 in which the next command to be issued can be derived by reusing either a history command or a task command.

<i>history command</i>		<i>task command</i>
		<code>ls ~/thesis/talks</code>
<code>^talks^proposal</code>	\Rightarrow	<code>ls ~/thesis/proposal</code>
<code>cd !-2:\$</code>	\Rightarrow	<code>cd ~/thesis/talks</code>
<code>cd !-2:\$</code>	\Rightarrow	<code>cd ~/thesis/proposal</code>

Example 2: A user intends to invoke the task command `cd ~/thesis/proposal`. It is a variation of an earlier task command which may be reused by typing a UNIX *csh* history command consisting of the word `cd` followed `!-2:$`. This history command retrieves the last parameter `~/thesis/proposal` from the second last task command (i.e., `ls ~/thesis/proposal`). However, it is the same history command used to generate the preceding task command, `cd ~/thesis/talks`. Another reuse operation, involving fewer keystrokes, is to invoke rather than retype the preceding history command by typing `!!m`. `!!m` is a meta-history command which repeats the previous history command.

Unaccepted user inputs are user inputs that are rejected by a system because they are entered incorrectly (e.g., misspelled command) or used in an inappropriate context (e.g., trying to examine a file which is not in the current directory). Current history tools consider these inputs as errors and therefore, irrelevant. However, as suggested by the use of history for error correction, users may expedite error recovery by correcting and/or reusing the unaccepted user input (e.g., correcting the spelling of a command).

Snapshots are visual snapshots of a system state. This information is important because human memory research indicates that people do not attend to or remember all the information with which they are presented [Lindsay & Norman, 1977]. Users may have ignored the information by mistake or thought that it is not important for subsequent interactions. Therefore, while the snapshots may appear to capture extraneous information, they allow users to examine, in retrospect, aspects of those snapshots that they did not attend to earlier or remember. Furthermore, the snapshots contain visual contextual cues which can help users recognize information that is difficult to recall.

Externalized Information and Internalized Information

As users perform tasks using computers, they mentally retrieve information (i.e., *retrieved* information) that is relevant to the performance of the task. One useful aspect of history tools is that they cache this retrieved information so that users need not retrieve it from their memory when it is needed again.

Our discussion of options for history information has focused on retrieved information that has been communicated to the system (i.e., *externalized* information). However, history tools should not artificially restrict the retrieved information to only that which the users have explicitly communicated. That is, history tools should also capture *internalized* information; retrieved information that users do not communicate to the system during a session.

Since internalized information is not explicitly made available in the course of the interactions, users must either volunteer the information or the system must probe the user for it. In the latter case, Ericsson and Simon (1980) claim that there are only two kinds of information that a system can reliably probe for: a) information in short-term memory (i.e., *information in focal attention*) or b) information in long-term memory for which the system can provide direct probes. They claim that intermediate results of automated processing¹ or ongoing cognitive processes (e.g., recognition processes, perceptual-motor processes, and long-term memory direct retrievals) are unavailable, but final results are.

User and Behavioural Information

The use of history for user modelling and user interface adaptation point out the need to maintain, in a user history, information about individual users and their behaviours. Such information includes user preferences, aliases, task preferences, and user behaviours. They are inferred from an analysis of the user's interactions.

This type of history information depends on the kind of information being sensed and the technique for interpreting and inferring knowledge and behaviour from the sensed information. As alluded to in Chapter 2, both the sensed information and the inference techniques are many and varied. They need to be determined in advance and incorporated into history tools. As an illustration, Chapter 4 describes a behavioural phenomenon observed in user interactions – the repeated reference to a group of previously issued commands. Chapter 5 describes a technique for isolating this behavioural phenomenon. Such a technique should be incorporated in history tools in order to support this behavioural phenomenon.

Basic History Components

Different history uses require different history processing capabilities. For instance, history for reuse, inter-referential I/O, and error recovery require tools to reuse relevant history items, to relate them to current interactions, and to make corrections. As well, history for navigation and reminding require the display of the history for consultation purposes. Similarly, history tools require additional capabilities for exploiting particular types of history use. For instance, reuse of a previous interaction with possible modifications requires editing tools to alter the selected history item so that it can be reused properly in the current context. Reuse of a functional group of prior interactions requires tools for grouping items of a history so that they can be reused as group.

In order to support all seven history uses, our analysis of the requirements of history tools for each of the seven history uses reveal the need for minimally supporting the following four components: *collection*, *presentation*, *submission*, and

¹ Such behaviours have become fully automated and the associated knowledge and intermediate processings are integrated and composed into high-level procedures which are executed without being interpreted [Anderson, 1982].

administration [Barnes & Bovey, 1986; Joy, 1980; Lau & Asthana, 1984]. The collection component captures relevant information for a user history; the presentation component displays a user history; the submission component allows users to select history information for use in the context of the current situation; and the administration component manages, internally, a user history.

Collection of History

Caching relevant information helps users to conserve valuable internal memory stores but may incur cognitive and physical overhead costs [Schonpflug, 1988]. These costs include internally encoding aspects of history information and extraneous processing in order to access and locate information in the user history. Also, while the availability of vast storage capacities permit the retention of a great deal of history information, there is concern for the economic and efficient deployment of computational resources. Hence, judicious selection of history information is important as it influences the usability and effectiveness of history tools. Issues concerning what to collect, how much to collect, and what means are used to collect the information are addressed, in part, by two strategies: *selective logs* and *user collection tools*. Note that neither strategy is considered wholly effective or efficient.

Selective Logs

Selective logs automatically collect information concerning a user's recent interactions that is directly accessible from information exchanged during the interactions. Information that is indirectly accessible can also be collected using appropriate heuristics which infer and extract this information. The heuristics are based on empirical observations of user interactions and history use, or knowledge about task and interaction protocols. A case in point is Greenberg and Witten (1988b)'s study which observed considerable temporal bias in the commands that are repeated; on the average, 60% of the repeated command lines occurred within the last 10 command lines. Thus, they suggest that a simple strategy would be to keep the last 10 command lines issued as they are most likely to be repeated.

User Collection Tools

Designers also recognize the fact that some information is difficult to identify and access, and suggest the provision of appropriate capabilities that permit users to designate and to cull such information. *User collection tools* provide annotation capabilities which allow users to volunteer internalized information. The annotations can enhance the information obtained in *selective logs* or fill in missing information (see Example 3).

An alternative annotation scenario is to log all meaningful input and output information from user interactions. This log is then supplemented with tools that would allow users to cull the information when needed. An example is the facilities available in some window systems like SUNTOOLS that save substantial

Example 3: A program management task involves a search subtask and program modification subtask. The search subtask involves scanning a hierarchical file system to locate source program files to be modified in the program modification subtask. Users may not be aware of the file search criteria. Therefore, they are unable to communicate this information as they sift through the file space looking for the desired files. An annotation tool would allow users to note the locations of the files so that they can be dealt with in the program modification subtask.

portions of old contents of windows. The contents can be reviewed with the aid of a scroll or search mechanism.

Presentation of History

The display of history is an important functional component of history tools. It augments user interactions by allowing users to access and use history information and provides an externalized representation of certain contents of a user's memory. As a clarification of terminology, the term *internal memories* refers to a user's own memories and the term *external memories* refers to any external media (e.g., notebook) which people use to record information from their internal memories. Two examples of external memories are visual stores and computer stores. Visual stores refer to a computer's display and computer stores refer to electronic files. The visual history mediates user access to external memories and facilitates certain mental and physical processing operations. Many of the current history tools, especially for reuse, are severely lacking in display support. Four different mental processing operations facilitated by a visual history and their associated support are examined:

- spontaneous reminding
- deliberate reminding
- visual scan and/or search
- problem solving and improvisations

Spontaneous Reminding

The old adage “out of sight, out of mind” is an apt introduction to a mental processing operation known as *reminding* which was first introduced in the discussion of history for reminding. A *spontaneous reminding* occurs when a user notices that an earlier situation shares certain similarities to the situation at hand [Ross, 1989]. Such phenomena may be nurtured by the presentation of a user history which provides *visual reminders of past experiences*. Also, by prominently displaying a user history, history tools help to augment a user's internal memories and to mitigate forgetting. Finally, the display of a user history leads to a user awareness of information that is not remembered or considered (i.e., it helps initiate reminding) as illustrated in the Example 4.

Example 4: A user has a vague recollection that a similar problem was previously encountered but does not remember the circumstances nor the solutions. In browsing a visual history, visual cues trigger the recall or recognition of relevant information and thereby help users to retrieve relevant knowledge about the problem and its solution.

Deliberate Reminding

When there are no visual reminders – from a lack of reminding information or visual representation schema – recall and recognition cannot be spontaneously facilitated. Then, the onus is on a user to initiate and pursue *deliberate reminding* by explicitly probing and extracting relevant history information. Such retrieval activities are fruitful if suitable cues are provided to elucidate the history information and to aid the retrieval of relevant history information.

There are cues that favour recognition mechanisms and those that favour recall mechanisms. The attributes that make cues effective are the subject of intense research and are particularly important if the potentials of history are to be realized [Larkin & Simon, 1987; Miyata & Norman, 1986; Reisberg, 1987; Schonpflug, 1988].

Visual Scan and/or Search

Recognition and recall operations may require that users perform a *visual scan and/or search* of their history. Many graphical systems exist today and they allow users to exploit the power of the display. A good graphical representation or diagram allows users to use quick perceptual judgements in place of abstractions and thereby expedite searches [Larkin, 1989]. Also, users may not have conscious access to scan and search knowledge, especially if perceptual knowledge is involved [Reisberg, 1987]. Finally, users may have difficulty in characterizing and communicating such knowledge because of limitations in interaction or communication capabilities. The display helps to bridge such difficulties by allowing a user to scan and search for the desired information.

Such visual processing operations are facilitated by a set of support capabilities. For example, users may need to locate and compare different pieces of history information which may be widely dispersed. A tool that allows users to locate and collect this information would be useful. Also, in order for users to identify and match a number of attributes, they need representational support to view and highlight information in different ways (e.g., fisheye views [Furnas, 1986]).

Problem Solving and Improvisations

Problem solving is often done in the context of an external display (Larkin, 1989 pg. 319). The display facilitates the process by reducing the complexity of the mental processes required. Furthermore, it is well-known that some people can think better with pictures because such thinking involves perceptual inferences as opposed to thinking with propositions which involves logical inferences

[Larkin, 1989]. As indicated in Chapter 1, *improvisations* occurring in a situation similar to one encountered earlier can benefit from access to history information from related situations. Also, a visual history can enhance problem solving. Thus, the display and a user history are possibly two of the many resources that users may use to support their problem solving or improvisations [Suchman, 1987].

Submission of History Item

One of the objectives of history tools is to reduce the physical effort required to specify commands after some form of the command has been previously externalized. The proposed benefits are increased speed, convenience, and economy of user actions as compared to that of retyping or repeating a user action. This type of support also frees the user from dealing with low level aspects of specifying commands or performing action sequences. The user is free to concentrate on the conceptual difficulties of a problem, especially when a similar problem has been previously encountered. This enables users to address difficulties associated with command specification and difficulties arising in the course of user interactions. In order to realize these benefits, the submission component must provide certain capabilities that will result in minimal distraction and minimal cognitive and physical effort. We propose four capabilities for facilitating submission of information from a user history: *identification*, *modification*, *retraction*, and *coordination*.

Identification

One important design requirement is to allow users to identify the desired history item for submission. There are two possible identification techniques: a) browsing a visual form of a user history to locate the history item and then pointing to it, or b) recalling the history item from a user history with a selection criteria. It is necessary to support both techniques as circumstances and individual preferences may call for the use of either technique. The concern is how to integrate the two techniques.

Selection of history information from a display is a natural use of the display. McMahan (1987) describes a mechanism which associates a type attribute to display objects and allows a particular interaction context to dictate the display objects that are selectable.

A history item is recalled when it is not visible, or it is difficult to locate visually, or it is more amenable to description than selection. The description may focus on certain distinguishing attributes of the item (e.g., a file name or a pattern occurring in the command) or on when the command occurred (e.g., last command). For example, in the UNIX *tcs*h filename completion mechanism, a user provides enough characters to identify a desired filename followed by an ESC character [Joy, 1980]. The facility completes the rest of the characters of the filename after consulting a list of filenames using the partial filename as a key.

Minimally, selection and recollection capabilities must be supported. A history type mechanism can be used to associate types to history information and to ensure that information appropriate to the current context can be selected or recalled. A history completion mechanism can be used to recall the particular

history information from a list of candidates provided by the history type mechanism. A facility integrating such a history type mechanism and history completion mechanism supports the selection and/or recollection of the desired history information from a user history. Such a facility is considered in Chapter 6 where a number of history identification schemes are analyzed.

Modification

Information in a user history may be reused for some other purposes, may have been entered incorrectly and require correction, or may be an appropriate substitute for the desired information. Thus, since the selected history information is not always in a desirable form, users need tools to modify selected history information into an appropriate usage form.

There are three important issues. First, it is rare for users to select and edit history information in one step, partly because of the cognitive and physical effort involved in juggling both operations. Thus, it would be desirable to separate the two operations so that a user can verify that the desired item is correctly selected before changes are made to it. This separation also helps a user to refine the selection criteria if the wrong history information is selected.

Second, the modification capability should support features from a user's preferred editor so users need not learn new editor commands. Thus, existing editing skills can be carried over to history tool usage.

Finally, users may need to make modifications which may affect different elements of a piece of history information (e.g., words, arrangements of words, meanings, concepts, intentions). Thus, an editor must facilitate this. The question is how does the modification component provide such functionality and encourage users to explicitly indicate which elements in a user history are used? If users are encouraged to make explicit the elements that are to be modified in the process of making the modifications, then the modification facility could assist the collection component in annotating the information in a user history.

Retraction

The ability to retract operations is desirable for error recovery purposes. Also, retraction is a desirable function of a history tool itself because inherent limitations of history collection facilities, such as difficulties in detaching graphical actions from context, may result in the erroneous use of a user history. In association with a coordination tool, retraction permits user improvisation and error recovery to be initiated and coordinated. The retraction tool may be as sophisticated as Archer, Conway, and Schneider (1984)'s tree-based UNDO, REDO, and SKIP.

Coordination

Information in a user history may be submitted by either a) identifying and editing the information or b) composing user inputs using information from a user history (e.g., ls !\$). The coordination capability is concerned with how history information is drawn into the course of user input composition. As suggested in

the section entitled “Contents of a User History”, there may be a number of different collections of a user history each maintaining different kinds of history information (e.g., locations of files, directories). In any usage situation, the desired information in a user history may come from one or more of these collections.

To support the formation of complex or compound commands, the coordination capability must allow users to group and generalize a number of history commands. The sophistication of this capability depends largely on the extent and the frequency with which users build complicated commands. The capability may be as sophisticated as a programming-by-example facility or as lightweight as a macro facility. However, it is reasonable to assume that the cognitive and physical effort involved in using the programming-by-example facility may limit a user’s desire to use such a facility. Furthermore, we noted in Chapter 1 that user interactions are error-prone, improvisational, non-systematic in their recurrences, and interleaved with other activities. This suggests that users would prefer finer control than a macro facility would allow. Thus, users may opt for a facility that allows them to sequence and queue up a number of commands from a user history prior to submission.

Administration of History

The fourth history component is concerned with the internal organization and preservation of history information. There are three areas of concern:

- Separating independent user histories
- Accessing history information
- Aging, saving, and discarding history information

Separating Independent Histories

In many of the history tools examined, there is little or no logical separation of collections of a user history (i.e., into input and output or into task-specific groupings). In general, a user history is associated with a user’s session. Such an ad hoc approach to maintaining a user history leads to a number of problems.

First, a user pursues a number of different activities concurrently within a session [Bannon, Cypher, Greenspan, & Monty, 1983; Cypher, 1986; Lee, 1992a; Miyata & Norman, 1986]. By grouping all the interleaved activities into a single collection, users may expend more time and effort to scan, search, and differentiate history information associated with a particular activity. As a result, they may be deterred by these inconveniences or they may be unable to capitalize on the benefits that the information in a user history provides. For example, users may not use their history to help them re-acquire the mental context for an activity because a great deal of effort is required to sift through their history to find the desired information.

It would be beneficial to maintain a user history appropriate to a particular set of activities, a work context or a task. Then, the issue becomes one of how users, in the course of their interactions, signify movement from one work context to another and thus associate subsequent activities to a particular work context or

task. As one solution to this problem, a simple windowing mechanism allows users to logically organize different tasks into different windows where activities conducted in a particular window belong to one task. Shifts in user attention or switches in tasks may be explicitly and effortlessly conveyed to a system in the course of a user's interaction [Bannon, Cypher, Greenspan, & Monty, 1983; Cypher, 1986; Miyata & Norman, 1986].

Second, the artificial separation of history by session is problematic. The end of a computer session does not necessarily herald the completion of a set of tasks but instead, for example, some other engagement may preclude continuation at the present time [Bannon, Cypher, Greenspan, & Monty, 1983; Cypher, 1986]. Similarly, when users remotely log into a computer from another computer, the remote session is not part of the local session. Hence, there is now a second user history (i.e., one on the remote machine and one on the local machine) containing information that is not easily accessible on the local machine.

Finally, some history information spans more than one work context (i.e., it is global) and should be available to other work contexts as illustrated in Example 5. However, it is not clear how such information is extracted and placed into a global user history.

Example 5: A UNIX user prefers to use the *ls* command with the *-F* option regardless of the particular user task. This command lists a directory's contents and marks items with an appropriate trailing symbol corresponding to the type of the item (e.g., a "/" for directories). Thus, this particular user preference should be kept in a global history rather than a user history for that task.

Accessing History Information

Occasionally, users may need to have access to history information that is older and/or not available on the display (i.e., it is in an electronic file). In such cases, they need capabilities that are akin to query and search capabilities which allow a subset of the history information to be extracted from the computer store for user perusal.

Aging, Saving and Discarding History Information

The final set of issues pertain to the management of history information in the display and computer stores. One display issue is how to *age* items in a user history so that items that are not referenced are less visible and less accessible. Also, the amount of screen real estate available to display a user history is limited and priority should be given to important history items. The issue of when and which items should be removed from the display is crucial because displacing important items may result in the loss of an item's service value to a user. These issues are reminiscent of the computer memory management issues for executing programs – a relationship which is explored further in Chapter 5.

One computer storage issue is how long to keep history information. Keeping information, that is never consulted or is ephemeral would needlessly consume

system resources and would obscure more important information. On the other hand, some information *persists* for some time, possibly indefinitely, and premature removal would render a user history ineffective. This includes temporary information that concerns a particular active task or permanent information, like important or novel task solutions and user preferences or methods for adapting system behaviour (e.g., Example 5). Information concerning an active task should remain until the task is completed. Thus, the persistence of a history item – ephemeral, temporary, or permanent – needs to be identified and appropriate storage actions undertaken.

Concluding Remarks

The design of an integrated history tool must be concerned with two important questions: what information should be kept in a user history and what functionality should be provided to support user access to the information in a user history? With these two questions in mind, our analysis of the seven uses of history tools led us to propose several answers (Table 1 summarizes our proposals).

While the seven uses of history can provide one possible source of insight to these two questions, additional corroboration and elaboration may be available from other sources. One source is naturalistic observations of user interactions which provide insights into behavioural characteristics and patterns evident in

Question	Options	Examples
Contents of User's History	Actions and objects of actions	full commands, file names, and directories
	System responses	selected state information
	States and transitions	snapshots of states and user actions
	Peripheral information	history commands, unaccepted user inputs, snapshots
	Internalized information	information in short-term memory and information in long-term memory that can be probed
	User and behavioural information	user preferences and behavioural phenomena
Basic History Components	Collection	selective logs, user collection tools
	Presentation	spontaneous and deliberate reminding, visual scan, and/or search, other mental processing operations
	Submission	identification, modification, retraction, coordination
	Administration	separate collections of a history, tools for internally managing a history

Table 1: Questions, options, and examples examined.

user interactions. These behavioural insights are the bases for heuristic techniques used by history tools to capture relevant information from user interactions. The studies in Chapter 5 are rigorous examinations of a behavioural phenomenon observed in a naturalistic study of UNIX user interactions.

Relevant history information may also be based on a theory of user interactions. For instance, Uejio, Blattner, Schultz, and Ishikawa (1991) describe two history tools developed from two different models of command retrieval. One command retrieval model suggests that users search for commands based on some semantic relationships, concepts, attributes, and abstractions. The corresponding history tool logs complex commands and maintains various attributes about these commands: frequency of use, files accessed, time executed, and keyword descriptions. The other command retrieval model suggests that, sometimes, users have a fuzzy recollection of the information. A neural network is used to maintain previous user commands and to allow users to recall particular history information using an incomplete specification of the desired information.

While behavioural studies and theories of user interactions can provide insights concerning useful history information and capabilities, they do not substitute for insights from a detailed study of user behaviour. Studying how users use prototype history tools or asking users about what history information and history functions they would like can be equally informative.

Chapter 4

User Interaction Behaviour and History Usage

The preceding analysis of the seven potential uses of history shed light on a number of cognitive and behavioural considerations underlying the use of history tools. This analysis permits a broad examination of the scope and coverage of these relevant variables and hence, user requirements. However, the insights are based on an informal task-artifact analysis of current history tools rather than a behavioural analysis of user interactions and history tool use. Behavioural analysis of computer-tool usage provides another source of information about user requirements; one that is directly supported by behavioural data. However, it is conducted within a specific and focussed context and is limited in the issues that can be examined and the generalizations that can be made.

This chapter describes an exploratory behavioural study of three UNIX command interpreters and their history tools. It is exploratory because everyday natural user interactions within UNIX are observed, unconstrained by experimental controls. The motivation for performing this particular study is presented, followed by a description of the study and its findings.

Behavioural Studies of User Interactions and History Use

Several studies have investigated how individuals and groups of individuals interact with computer systems. Most of them concentrate on command usage and makeup of command sets to derive general implications for user interface design (e.g., Akin, Baykan, & Rao, 1987; Boies, 1974; Draper, 1984; Greenberg & Witten, 1988a; Hanson, Kraut, & Farber, 1984; Krishnamurthy, 1987). They identify important behavioural trends regarding frequency of command use or growth in command sets. However, very few studies have focused on uncharacteristic user interaction behaviours arising from constraints in computer tools and environments (i.e., behavioural phenomena). An example of such a study is Greenberg and Witten (1988b)'s study which found that users frequently issued previous command lines (i.e., on average, 74% of the commands lines are recurrences). We are interested in such behavioural phenomena because they can provide insights into the kinds of user support tools, particularly history tools, that we need to provide in order to nurture and enhance user interactions.

Furthermore, in order to shed light on how history tools can provide user support for user interactions, we need to examine the kinds of functions in history tools that people use. We are aware of only one study that has examined actual usage of history tools (i.e., Greenberg and Witten (1988b)'s study). That study documented the extent of history usage (54% of the subjects used history but in only 4% of their commands) and the referenced history items (the bulk of the references were to the last 2 commands), but their study did not examine the history functions that were used. Such observations could reveal the strengths and weaknesses of history tools and the user needs and requirements with respect to the content of a user history and the history functions.

Exploratory Study of User Interactions and History Usage

Since previous studies have not examined certain behavioural aspects related to user interactions and history-tool usage, we conducted our own study. The study has two objectives. The first objective was to identify behavioural phenomena in everyday natural user interactions that are particularly relevant to history-based, user support tools. The second objective was to examine the history functions that users employ and to compare usage of two history tools.

The exploratory study observed everyday natural user interactions of users of three different UNIX command interpreters – Bourne Shell (*sh*) [Bourne, 1979], C Shell (*csh*) [Joy, 1980], and TC Shell (*tcsch*) [Ellis, Greer, Placeway, & Zachariassen, 1987] – with no experimental controls and no observer present. Three UNIX shells are examined because they provide a richer pool from which subjects can be drawn and from which similarities and differences in user interaction behaviours and history use can be observed. While this methodology may reveal natural behavioural patterns evident in user interactions, it does not provide information about the factors influencing the behaviour. Therefore, its findings should be interpreted with caution. Furthermore, UNIX is a textual, command-based system and the contents of a user's history in *csh* and *tcsch* are the command lines. Therefore, issues pertaining to graphical, gesture-based systems are not explored.

A possible criticism may be made about studying a system with poor usability such as UNIX when direct manipulation systems like the MACINTOSH are popular and widely used. The following reasons are offered in answer to such a criticism:

- (1) There is a large user community with varying skills and task complexities.
- (2) This study adds to the accumulating body of knowledge about user interactions in UNIX [Akin, Baykan, & Rao, 1987; Bannon, Cypher, Greenspan, & Monty, 1983; Desmarais & Pavel, 1987; Draper, 1984; Greenberg & Witten, 1988a; Greenberg & Witten, 1988b; Hanson, Kraut, & Farber, 1984].
- (3) The shells permit user access to a wide range of UNIX applications. Hence, we can potentially observe general patterns of behaviour.
- (4) Most UNIX shells support history tools (e.g., Korn Shell, C Shell, TC Shell).

While UNIX has some usability problems which may result in some artificial behaviours, these problems should not preclude the observation of behaviours that are fundamental to user interactions with textual command-based systems.

Method

UNIX Shells and History Tools

A *UNIX shell* is a programmable UNIX command interpreter. We selected *sh*, which does not support a history tool, because user behaviours in this shell can be compared against those behaviours observed in UNIX shells which support history tools (i.e., *cs*h and *tc*sh). In *cs*h and *tc*sh, command lines that are issued are collected into a *history list* which is displayed upon the user's explicit request. The history list is a user history and each of its entries contains a user command line known as an event.

A *cs*h user command line may be a shell request (e.g., `ls file`), a history request (e.g., `!!`) or a hybrid request (e.g., `!! | lpr -Pntx`). A hybrid request combines shell and history requests. A *cs*h history request has the following syntax:

event designators [: *word designators*] [: *modifiers*]

Optional items are delimited by the symbols `[]`. *Event designators* select particular events in a history list (e.g., `!!` for the most recent history event). *Word designators* select words from the associated history event (e.g., `:2-4` for the 2nd, 3rd, and 4th word). *Modifiers* request that changes be made to the associated history event (e.g., `:1` extracts the second word of the associated event).

This syntax allows event designators to be bound to zero or more word designators and/or modifiers (e.g., `!-5:s/aaa/bbb` recalls the fifth last event and makes the substitution before submitting the modified history event). Thus, users can modify a history event before issuing it. *cs*h provides a number of short cuts for word designators (e.g., `!$` is the last word of a history event) and modifiers (e.g., `~pattern~replacement` replaces portions of the last event matching *pattern* with *replacement*).

The command *history* prints a portion of a history. As well, when the `.p` modifier is used with a history event designator, the retrieved history event is printed rather than issued (e.g., `!!:p` prints the last history event). Table 1 lists the number of event designators, word designators, modifiers, and history print requests in *cs*h. Note that the number of modifiers excludes the `.p` modifier which is accounted for in the number of history print requests.

*tc*sh is a superset of *cs*h and supports two enhancements to the *cs*h history tool: a) a history retrieval operation may be performed separately from a history modification operation and b) command line editing and history manipulations are facilitated by emacs-style control keys.

Event Designators	Word Designators	Modifiers	List Contents of History	Total Number of Operators
7	11(4)	9(2)	2	35

Table 1: The number of *cs*h history functions in each category. The number in a parenthesis represents the number of short cuts.

Two history retrieval options are available in *tcsht*:

- a) step backwards and forwards through a history list examining each event one at a time, using a cursor-like control key.
- b) provide a pattern followed by a search backwards/forwards control key. The system searches backwards or forwards through the history list for the first occurrence of a history event beginning with the characters in the pattern.

Both history retrieval options and edit operations are invoked from the user's current input location in the shell session with the retrieved history event overwriting the history retrieval request. Effectively, a one-line edit buffer exists at the user's current input location.

Data Collection

The Berkeley 4.2BSD *script* program records everything printed in a user session. When it is invoked, its process is inserted between the shell process and the terminal process. The terminal process controls user inputs from the terminal's keyboard and system outputs for the terminal.

The *script* program was modified to produce a *server* and *client* version of the program. There are two reasons for doing this rather than modifying the three shells. First, less effort is required to instrument one program than to instrument three UNIX command interpreters to log the usage data. Second, using this approach, it is possible to identify when users invoked history functions, what sorts of history functions they made use of, and what sorts of command line editing were performed. Logging the data after the user's input is parsed by the shell precludes access to the history usage data.

In the *server-client* version, the *server-script* process rendezvous with one or more *client-script* processes dispersed, possibly, on different machines to record the data collected by the *client script* program into a central file. Each *client-script* process is associated with a user session and records the raw keystrokes (e.g., vi [^]H !S) initiated in that user session.

In general, a shell process parses the raw keystrokes to produce the fully-parsed command line. If we only capture raw keystrokes, we have no access to the fully-parsed command line. We can interpret some of these raw keystrokes (e.g., the raw keystrokes *cd ^H Hls News* can be easily interpreted as *ls News* because [^]H is the control character for backspace). However, we are unable to interpret all raw keystrokes. For example, we cannot interpret the [^][character in the raw keystroke *cd News/ ^[* without recording information about the contents of the directory from which this command was issued because the control character [^][completes the file name with the name of a file in the current directory matching the starting pattern.

Since we cannot interpret all raw keystrokes without recording extraneous information, we use a heuristic to capture fully-parsed command lines that the shell echoes as it parses raw keystrokes. This heuristic scans the first *n* characters of output lines from the shell process for the user's prompt character *c*. The two parameters, *c* and *n*, are set up individually in consultation with each subject at the beginning. We had mixed success using this heuristic to record every fully-parsed command line. Consequently, we did not manage to record all fully-parsed

command lines matching all recorded keystrokes. While this flaw is a drawback, it is not a serious one because we are able to observe, qualitatively, some behavioural phenomena based on the fully-parsed command lines that we did record. However, we are unable to quantify the extent of any observed behavioural phenomena (e.g., mean number of recurrence of the same fully-parsed command line).

Subjects

6 *sh*, 10 *cs**h*, and 7 *tc**sh* users volunteered for the study. They were graduate students, staff, and faculty members from computing sites on campus who were skilled users of the particular shell. They used UNIX to carry out their day-to-day tasks (e.g., system administration, programming, statistical analysis, document preparation and communication).

Knowledge and use of the history tool were not prerequisites for participation in the experiment as the study's objectives were to get a representative sample of users. If such prerequisites had been used, the study would have ignored the behavioural characteristics of non-history-tool users and we would not have gotten a sense of the number of non-history-tool users.

In the analysis, only the data for those subjects who issued 500 command lines or more during the monitoring period were examined. This restriction eliminated those subjects who decided to withdraw from the study and allowed us to observe behavioural patterns in UNIX user interactions. In the latter case, any behavioural patterns existing in user interactions would be more evident in sessions where many user command lines were issued. As a result, only the traces for 3 of the 6 *sh* (S1-S3), 5 of the 10 *cs**h* (S4-S8), and 5 of the 7 *tc**sh* subjects (S9-S13) were analyzed.

Procedure

Each subject ran the client-script program in each of their shell sessions. The program timestamps and transmits all the user's raw keystrokes to the *server-script* process. Subjects were monitored in their natural work setting over a one week period with no experimenter present. The following data were collected: 1) the user's raw command line keystrokes, 2) the parsed shell command line, and 3) the time that it took the user to issue the command line.

Results

The findings are divided into three parts. Part one discusses a basic property evident in all the subjects' command lines and parts two and three discuss the history functions that *cs**h* and *tc**sh* subjects used.

Three Behavioural Patterns

There are three patterns evident in our subjects' traces (see Table 2):

- Users *repeat* individual command lines or their parameters.
- Users *cycle* through one or more command lines.
- Users *group* command lines issuing them in no specific order.

As indicated earlier, because we are unable to capture all fully-parsed UNIX command lines, we are unable to characterize the frequency of each behavioural pattern (i.e., fraction of the session where each pattern was observed). However, this problem simply means that we cannot provide this statistic.

Parameters of command lines are not the same when they are re-issued. Observations and user comments suggest that these patterns are the result of task and user factors. Certain tasks are inherently repetitive while others are complex and ill-defined and require a trial and error approach. Users make errors and must make corrections and adjustments to their actions (e.g., misspelling or mistyping the command line) or they are unfamiliar with the system or the task.

History-Tool Functions Used by csh Subjects

Table 3 summarizes the extent to which *csh* and *tcsh* subjects, respectively, used the various functions available in their history tools. Each figure in the table

Description of Pattern	Example
repeating command lines and parameters literally	spell paper.tex page spell paper.tex page
repeating command lines with a different parameter	rlogin utworm rlogin utbugs
repeating with the same parameter but a different command name	page draw vi draw
cycling through multiple command lines with different parameters	setenv TERM xterm gnuemacs -e mh-r setenv TERM vt100 gnuemacs -e mh-r setenv TERM sun gnuemacs -e mh-r
grouping command lines and executing them together	S < draw > draw.out page draw.out page draw.out vi draw S < draw > draw.out vi draw

Table 2: Examples of repeating, cycling, and grouping.

represents the percentage of the command lines the subject issued during the monitoring period that were issued using a particular history function¹. All *cs*h and *tc*sh subjects, including those who do not provide a long enough trace, made some use of the history tools.

Subjects did not, in practice, use the full range of functions available in their history tools (see Table 3) because they had limited knowledge of the repertoire of history functions. Furthermore, only a small fraction of the subject's session involved history-tool use. Subject S9 used a history tool the most (14%) while subject S8 used it the least (3%). Since we did not have access to all fully-parsed command lines, we are unable to compare the number of command lines that were literal repetition of earlier command lines against the actual number of command lines formed using history tools. However, this simply means that we cannot report this statistic.

*cs*h subjects tend to use simple history functions involving literal recall of a history event (i.e., *!!* for last event and *!pattern* for most recent event beginning

History-Tool Function Used	Usage (% command lines)									
	<i>cs</i> h Subjects					<i>tc</i> sh Subjects				
	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
<i>cs</i> h event designators										
!!	1.62	2.28	0.26	0.39	0.70	0.29	0.59	-	0.19	-
!pattern	3.28	5.52	6.78	2.42	0.53	0.75	2.58	3.06	6.74	0.63
!number	1.18	-	-	-	-	-	-	-	0.19	-
!-number	1.36	0.24	-	-	-	-	-	-	0.19	-
!?pattern	0.39	-	-	0.06	-	-	-	-	-	-
<i>cs</i> h word designators										
!\$	3.81	2.16	1.92	0.29	1.23	1.38	-	0.74	-	0.32
<i>cs</i> h modifiers										
!! + add on	1.09	1.32	1.79	0.35	-	0.12	-	0.37	0.19	0.63
:gs/aa/bb, :s/aa/bb	0.07	-	-	0.04	-	-	-	-	0.10	0.16
^pattern^replacement^	0.96	0.36	0.13	0.33	-	0.12	-	-	0.58	-
<i>cs</i> h requests										
list history	-	1.75	-	.16	.18	1.21	-	-	0.96	-
<i>tc</i> sh enhancements										
step back/forth						8.19	9.61	5.76	0.19	2.22
search back/forth						2.14	0.20	0.25	-	0.63
Total	13.76	13.63	10.88	4.04	2.64	14.20	12.98	10.18	9.33	4.59

Table 3: History-tool functions used by *cs*h and *tc*sh subjects.

¹ Our figures are larger than Greenberg and Witten (1988b)'s figures because they may reflect situations in which a command line is composed using a number of history functions. Thus, the reader should be cautious about making direct comparisons between the figures from the two studies.

with the given pattern). Some subjects did make minor use of other event designators: *!n* – an absolute reference to the *n*th history event, *!-n* – a backward relative reference to the *n*th history event, and *!?pattern* – a reference to the most recent history event containing a given pattern.

In a number of cases, the subjects included words from a history event in composing their command lines (e.g., *string * | !bm* takes the output of *string ** as input to a previous command line beginning with the command *bm*). The principal reference was the short-cut word designator for the last word of an event (i.e., *!\$*). A few of the subjects made rare use of the other *word designator* functions: both the regular form (i.e., *:I*) and the short-cut form (i.e., *!**).

A few subjects opted to make modifications to a history event before issuing it (see the *cs*h usage figures for the *cs*h modifier in Table 3). In many of these cases, subjects either added text to the end of the recalled event (e.g., *!! -I* invokes the last event with a *-I* option) or changed the previous event using the short-cut substitution operation (i.e., *^pattern^replacement*). Some subjects did make use of substitution requests available in the *:modifier* function; they included the *:s/aa/bb* request which replaces an occurrence of *aa* with *bb* in the associated event and the *:gs/aa/bb* request which replaces all occurrences of *aa* with *bb* in each word of an event.

Finally, only 2 of the 5 *cs*h subjects used the history print function.

History-Tool Functions Used by tcsh Subjects

tcsh subjects made slightly greater use of history tools compared to *cs*h subjects. In general, they preferred to use *tcsh*'s visual history *step back/forth* rather than *cs*h's *event designators* to recall a history event. However, most *tcsh* subjects opted to make use of *cs*h's method of recalling an event when they were searching with a pattern (i.e., *!pattern*) rather than the equivalent *tcsh* request, *search back/forth*.

tcsh subjects made less use of *cs*h event designators and word designators; in general, they used the same simple functions that were used by *cs*h subjects. *tcsh* subjects preferred to use control keys to edit command lines. The two rows of figures in the *tcsh enhancements* section of Table 3 bundle the cases when *tcsh* subjects simply recalled a command line and when they recalled a command line and modified it with control keys. However, *tcsh* subjects did make modifications when they recalled a history event which is unlike the *cs*h subjects who tended to recall the history events literally. While modifications were performed, they were very simple ones. Typically, *tcsh* subjects altered a few characters at the end of the recalled command line or altered a word in the command line. *tcsh* subjects did not make changes to more than one word of the recalled command line.

Discussion

Our study is only one of two studies to examine the nature of history tool usage; the other was by Greenberg and Witten (1988b). While their study provides a number of useful findings and design guidelines, we need further elaboration, corroboration, and investigation of history tool usage. Also, our study examines

the types of history functions that are used, in practice, while their study quantified the extent to which history tools are used without analysis of the functions used.

The findings should be interpreted with caution because the study is not controlled, the subject pool is not large, and only command-based interfaces are studied. However, there are several implications for user requirements.

First, the observed patterns – systematic repetition of whole or parts of one or more command lines – and the nature of history tool usage provide evidence of reuse of whole command lines as well as partial reuse of commands and arguments. Thus, the ability to refer to components of a history event is desirable.

Second, subjects made changes to some recalled command lines before issuing them. The changes were made with substitution requests available in *cs*h and emacs-like edit commands in *tc*sh. Thus, the ability to make changes to recalled command lines is desirable.

Third, subjects used a small fraction of the 35 different history functions available in *cs*h. This observation is explained, in part, by the subjects' limited knowledge of the history functions. Thus, user training is desirable.

Fourth, subjects did not always use history tools to re-issue previous command lines. Greenberg and Witten (1988b) suggest that this is because the syntax for recalling a *cs*h history event is obscure and arcane. This is substantiated by the fact that, although the *cs*h history functions are available to *tc*sh subjects, 4 of the 5 *tc*sh subjects prefer to use the *tc*sh enhancements (see Table 3). These enhancements permit users to manipulate history events directly and this function may be more desirable than all the *cs*h functions for recalling and modifying history events. *cs*h provides an abstract and symbolic means of selecting and editing history events (e.g., use of regular expressions and pattern matching) compared to the direct interaction with history events that *tc*sh provides.

Fifth, the lack of a permanent display of a user history may be another plausible reason for the minimal use of history tools. Visual presentation of history events is important both for refreshing a user's memory and for allowing a user to consult the history. Neither history tool provides a permanent display of portions of a user history except upon explicit user request. The history usage data shows that both groups of subjects make very few explicit requests to display events in a history list (see Table 3). As such, *cs*h subjects must rely heavily on their memory of recent history events. The task of recalling a history event in the absence of any external support from the display is memory-intensive and users may opt to selectively remember and recall memorable history events (e.g., recent command lines). In contrast, *tc*sh users are able, at least, to peruse history events as they single step through the history list.

Concluding Remarks

There are two observations from the exploratory study. First, there are three user interaction patterns: repeat individual command lines or their parameters, cycle through one or more command lines, and repeat several command lines in no specific order. These patterns reveal a property evident in all the subjects' traces; subjects repeat command lines, in whole and in part, as a single directive and as a

group of directives. While Greenberg and Witten (1988b) observed that individual command lines are repeated, our study observed periods in a user's session in which the repetition is made with respect to a collection of command lines (see Table 2). The extent to which this property occurs in a subject's session is not quantified because it was not always possible to deduce the complete command line that users issued. In the next chapter, this behavioural phenomenon is detected mechanically and the extent to which it occurs is characterized for a different but larger set of trace data, using the concepts of locality and working set from computer memory research.

Second, some of the commands lines that the subjects re-issued were submitted with the aid of a small subset of the history functions. Of the *cs*h history functions used, the bulk of them were simple ones such as the literal recall of a history event or the use of short-form history functions. By and large, *tc*sh subjects recalled history events by stepping or searching back and forth through their history and rarely made use of the *cs*h event designator functions. However, unlike *cs*h subjects, *tc*sh subjects edited command lines retrieved from their history list, changing a few characters or a word in the history event.

These two observations suggest three user requirements. First, users need better system support to facilitate the repetition of previous command lines (i.e., a history tool for reuse). Second, users need to be able to refer to components of a history event, to make changes to the recalled history event, and to display their history list. Third, users need to be aware of the availability of history tools and their functions.

In light of the limited set of capabilities available in current history tools (see discussion in Chapters 2 & 3), it is not surprising to discover that subjects do not make more extensive use of history tools when the opportunity arises. Besides poor design and poor user knowledge of history functions, there are two other factors that can influence people's use of history tools: availability of suitable reuse candidates and user effort required to use history tools.

The first factor is the ability to characterize relevant history events and to predict opportunities for using history tools. One such type of opportunity is suggested by the three command-line reference behaviours observed our exploratory study. The next chapter is a systematic examination of the three behavioural patterns and the implications of these reference behaviours on the prediction of history-for-reuse opportunities and history-for-reuse candidates.

The second factor is the mental and physical effort associated with the use of history tools. History tools may impose undue mental effort as compared to the physical effort required to retype a command line. As indicated earlier in Chapter 1, users want to minimize such mental effort. The tradeoff between mental and physical effort associated with using or not using history tools is examined in Chapter 6.

Chapter 5

Locality in User Interactions

In the context of human-computer interaction, *recurrence* is the phenomenon in which prior user interactions (i.e., actions and objects of actions) are referenced repeatedly in the course of a user session. One perspective of recurrence is known as *recency*. Recency is the phenomenon in which recent, individual, user interactions are referenced repeatedly during a user session; Greenberg and Witten (1988b) observed that recurring UNIX command lines were 1 to 3 command lines apart.

Another perspective on recurrence in user interactions was identified in our earlier exploratory study. During certain intervals of a user session, a user references repeatedly a small and related set of user interactions (see Table 2 in Chapter 4). This clustering of user interactions is nominally referred to as *locality* because it is akin to a behaviour by the same name observed in program memory references. Unlike recency, which characterizes recurrences in terms of the distance (i.e., references to individual user interactions that are close to each other in the history), locality characterizes recurrences in terms of periods in time where references are made solely to a small group of user interactions¹.

This chapter describes the application of computer-memory-research techniques and findings to examine locality in user interactions. First, does locality exist in user interactions and to what extent? Second, is locality a randomly occurring behaviour or is it an artifact of user interactions? Finally, does locality explain Greenberg (1988b)'s observations of history usage and the performance of history prediction.

Program Memory Reference Research

As a program executes, it makes references to portions of a computer's memory in units known as *segments*; a sequence of memory references is a *memory reference string*. It is neither feasible to allocate all of a computer's memory to a program nor efficient to allocate too much to a program since it penalizes other

¹ There can be recency without locality. However, if there is locality, then there is also recency.

simultaneously executing programs. Denning (1980) proposed a nonlookahead strategy which assigns memory to those portions of a program's memory space that will be needed immediately, based on sampling past references. This memory policy evolved from two concepts: *working sets* and *locality*.

Working Sets

A *working set*, specifiable as a program executes, is the set of segments that were referenced by an executing program within the last T references [Denning, 1980]. It is the basis of the nonlookahead *memory policy* proposed by Denning (1980) in which a program's working set is always kept in resident memory. Figure 1 illustrates a working set consisting of 7 segments { A, B, C, D, E, F, G } for a working-set window size of $T = 10$. The working-set definition suggests a process for dynamically estimating segments currently needed by a program; the process involves sampling past references using a moving window of size T .

The working-set concept has led to a class of tools and procedures for measuring and calculating intrinsic memory demands of programs. These tools and procedures have helped researchers to understand and to model program behaviour [Denning, 1980]. One such tool is the *inter-reference distribution* which is used to calculate the rate of segment faults. Given a reference string and a segment s , the *inter-reference interval* d is the distance between two successive references to s . An *inter-reference distribution* is the probability that two successive references to a segment are d units apart (see Figure 2). A segment fault occurs when successive references to a segment are made outside a sampling window of size T (i.e., $d > T$). The nature of this distribution may suggest interesting behavioural characteristics. Specifically, when the distribution is skewed towards low inter-reference intervals, it illustrates a strong *recency* effect and hints at a possible *locality* effect. For the memory reference string used in Figure 1, Figure 2 contains its inter-reference distribution which is skewed to smaller distances d .

Locality

Locality is the phenomenon in which a program's memory references, made in the course of the program's execution, are limited to a small subset of its virtual address space for an extended time interval. This behavioural phenomenon was observed in studies of program behaviour and is akin to behaviour observed in user interactions. That is, user interactions exhibit a temporal and spatial bias; there are periods in a user's interactions where a user repeatedly references a small subset of the user actions appearing in the user's session.

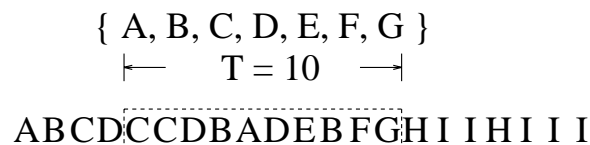


Figure 1: Working set of size 7 with a window size, T , of 10.

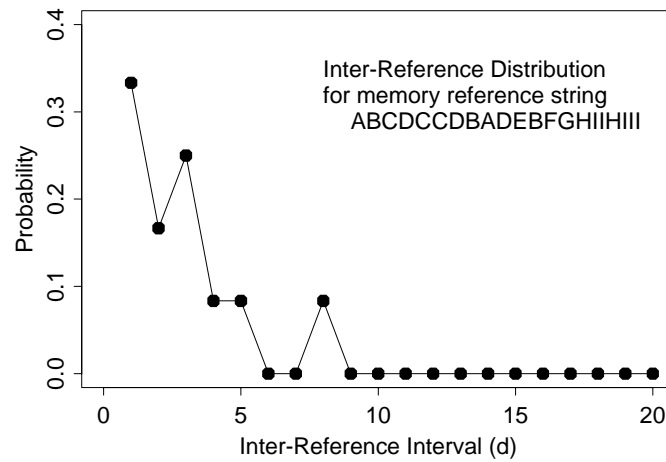


Figure 2: An inter-reference distribution for the memory reference string used in Figure 1. The distribution plots the probability that two successive references to a segment are d units apart. In this example, the distribution is skewed to small inter-reference intervals d illustrating a *recency* effect.

The locality phenomenon was formalized into a program memory reference model known as *phases* and *transitions* [Denning, 1980]. Phases are intervals in a program's memory references where locality is exhibited. A *locality set* is the set of segments referenced within a *phase*. Transitions are intervals in a program's memory references where a locality set is changing from one set to another.

While the phases and transitions model formally characterizes locality, it does not precisely define what constitutes an interval of distinctive referencing behaviour (i.e., a phase). Denning (1980) enumerates a number of precise definitions of locality each of which are based on a different criteria for identifying a phase. Madison and Batson (1976) provide one such definition and an algorithm for classifying a phase.

This locality-detection algorithm, sketched in Appendix B, makes use of a least recently used (LRU) stack. As segments are referenced, they are pushed onto the top of the LRU stack. A set of segments of size l is formed when the l segments occupy the top l elements of the LRU stack. A set of segments is a *locality set* if and only if all its members are each referenced at least once after the set is formed and no segments are referenced which are not in the set. Its *phase* is the maximal interval within a program's memory references in which all references made after the formation of the locality set are only to segments in the locality set. The phase begins with the reference to the newest member of the locality set and ends with the reference prior to a reference to a non-member of the locality set (i.e., minimal interval length p is $p = l$ where l is the locality set size). The interval preceding a phase, known as *phase formation* (see Figure 3), is the interval where the first reference to the oldest and newest member of the locality set is made (i.e., interval in which the locality set is being formed).

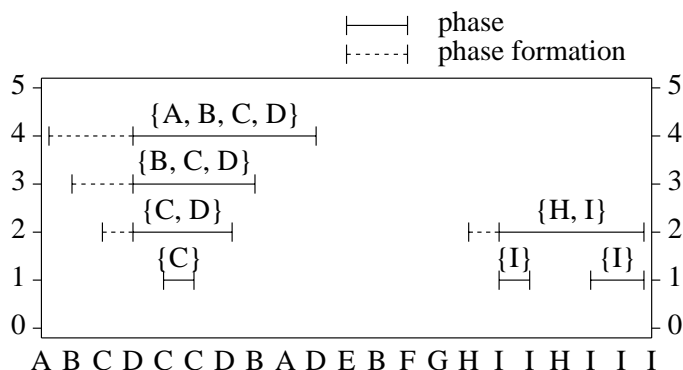


Figure 3: Locality sets of various sizes for the reference string used in Figure 1.

Unlike working sets, phases and locality sets are not determined simply; a locality set is not detected until each member of the set is re-referenced after the set is formed (i.e., at some point within a phase). Figure 3 illustrates an example of the compositions and sizes of all the locality sets appearing in the reference string used in Figure 1.

There are two reasons why a *working set* is not a *locality set*. First, a working set captures the set of segments within the last T memory references. It may include segments appearing in phases as well as transitions. Compare the working set for window size $T=10$ with the locality sets appearing within the same working-set window in Figure 4. Second, the process for determining working sets does not differentiate phases from transitions and does not identify boundaries for phases and transitions (compare the phase diagram with the working set in Figure 4) [Henderson & Card, 1986a].

Ideally, in the absence of any characterization of the extent of locality in program memory references and the phase lengths, a memory management policy should select the locality set as the resident memory set. During transitions, where references are typically erratic, no nonlookahead memory management policy, including working set, can properly anticipate the relevant set of segments.

Relevance to the Study of Recurrences

Studies of locality of program references lead to two insights about optimal memory management which are applicable to the study of recurrence in user interactions. These two insights led computer memory researchers to conclude that the working-set policy, with an appropriately chosen window size T , was superior to all other memory policies.

First, researchers found that local memory management policies were more effective than global memory management policies² [Denning, 1980]. The reason is that a local memory management policy, like a working-set policy, is more

² Local memory policies take into account the behaviours of individual programs while global memory policies focus on the aggregate behaviour of all active programs.

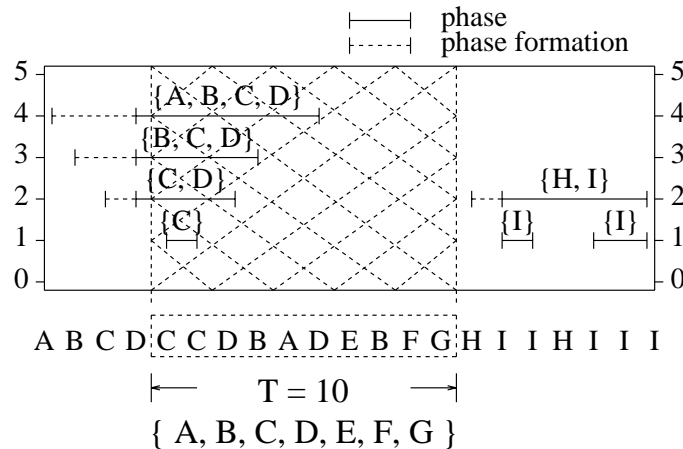


Figure 4: This figure combines Figures 1 and 3. The dashed region delimits portions of the phase diagram and the memory reference string covered by the sampling window of size $T = 10$. For the given sampling window, the working set is $\{A, B, C, D, E, F, G\}$ and the largest locality set appearing in the sampling window is $\{A, B, C, D\}$. While the working set captures four segments belonging to four of the locality sets contained in the sampling window, it also includes three segments $\{E, F, G\}$ which are not part of any locality set. Furthermore, the last four memory references EBFH are to segments appearing in a transition.

sensitive to changes in memory references of individual programs and can offer a much finer level of adjustment. In light of this result, it would be useful to examine command recurrences and methods for predicting recurrent commands which pay attention to a user's interactions.

Second, empirical data on program locality revealed that programs exhibited good locality. This means that a large percentage of virtual time (over 90%) was covered by long phases (consisting of 10^5 references or more to segments). [Denning, 1980]. Since the time spent in a phase was long and the mean time between two references to the same segment was short, a program's working set provides a good approximation of a program's locality set; it is effectively a program's locality set. That is, when programs exhibit good locality and phases are long, a working set captures primarily segments belonging to a phase and rarely segments belonging to a transition. It is because of these two phenomena – good locality and long phases – and because working sets are easy to determine compared to locality sets, that a program's working set provides a good approximate measure of a program's memory demands as well as a good memory policy for selecting segments to keep resident in memory. In light of this observation, it would be useful to find out the extent to which user interactions exhibit locality and the appropriateness of the working set as a policy for predicting recurrence candidates in user interactions.

Previous HCI Attempts to Apply these Concepts

Analogous to the computer memory management situation, there are HCI situations where it is necessary to identify a small subset of relevant items. As such, there have been two previous HCI efforts to employ these concepts.

Inter-reference distribution, working set, and locality were used to demonstrate that user references to multiple graphical windows exhibit locality and recency [Card, Pavel, & Farrell, 1984; Henderson & Card, 1986a]. While these results are encouraging, they are by no means definitive. One reason is that the study was not an in-depth formal analysis of window reference behaviours. Another reason is that locality in window references does not necessarily suggest locality in other user interaction behaviours (e.g., command line references). Finally, unlike command line references, window references involve a coarser grain of behaviour and a conceptually higher level of user interactions. Further study of locality for a wide range of user interactions is needed.

Inter-reference distribution was used by Greenberg and Witten (1988b) to analyze user interaction behaviours. They measured the frequency of command line recurrences as a function of the inter-reference distance d and found that the resulting inter-reference distribution was skewed to smaller distances. This finding, coupled with the fact that the average recurrence rate was 75%, provided evidence of *recency*. On the basis of this evidence, Greenberg and Witten (1988b) adopted the working-set strategy as the basis for predicting reuse candidates and examined its performance when different arrangements of the reuse candidates in a user history are employed.

Importance of Considering Locality

While both studies shed light on the applicability of the two concepts (i.e., working sets and locality) to HCI, there are two unanswered questions pertaining to locality in user interactions.

First, does locality exist in command line references and if so, is it a randomly occurring behaviour or is it an artifact arising from user interactions? There is evidence to suggest that locality is exhibited in graphical window and command line references (see Card, Pavel, and Farrell (1984) and Henderson and Card (1986a), and our exploratory study) but there has been no formal study of locality in user interactions. Recurrences in user interactions arise because of 1) the inherently repetitive nature of certain user tasks, 2) the nature of and difficulties encountered in user interactions (e.g., engaging in multiple activities, improvisations, and error corrections), 3) the inflexibilities in user interfaces which constrain the user's choice of actions, and 4) the bias toward the path of least cognitive resistance. We claim that:

Claim 1: Recurrences, and in particular locality, in command line references is not a randomly occurring phenomenon but a behavioural artifact arising from user interactions.

Second, is locality a more useful characterization of recurrence than recency? We raise this question because of two puzzling results from Greenberg and Witten (1988b)'s study of the recency phenomenon in command line recurrences. The two puzzling results are: 1) the high recurrence rate (i.e., 75%) but low history usage

rate (i.e., 4%) and 2) the suboptimal performance of the working set strategy for predicting recurrent commands. These two findings were based on a study of the recency phenomenon in command line recurrences. In the following sections, we explain how both results can be clarified if locality is considered.

While difficulties in using history tools offer one possible explanation for poor history usage, another explanation is that the recurrence rate and the history usage rate are not estimates of the same behavioural phenomenon and it is incorrect to correlate them. The recurrence rate R is the proportion of activities in a session which occur two or more times [Greenberg, 1988b]:

$$R = \frac{\text{number of activities} - \text{number of unique activities}}{\text{number of activities}} \times 100\%.$$

where *number of activities* is the total number of activities in a session (for the reference string shown in Figure 1, $R = (21-9) \times 100\% / 21 = 57\%$). An activity is either a command line or a command name associated with each user interactions. The UNIX *cs*h history tool is primarily a reuse tool and the history usage rate represents those reuse opportunities which are detectable as a result of the use of the history tool. When we associate recurrence rate to history usage rate, we are incorrectly associating a number of different user intentions (e.g., error recovery and others presented in Chapter 2). This could lead to an erroneous association of the recurrence of a command line as one reason for recurrence (i.e., reuse). Note, this is not to suggest that the recurrence rate is uninteresting in and of itself. However, we need a refined measure of Greenberg (1988b)'s recurrence rate R which is a better explanation of the reuse phenomenon and is consistent with the observed history-for-reuse usage rate. $R_{locality}$ is such a recurrence rate and represents the extent of locality in user sessions:

$$R_{locality} = \frac{\text{number of locality activities}}{\text{number of activities}} \times 100\%$$

where *number of locality activities* is the number of activities in a session appearing in phases (for the reference string shown in Figure 1, $R_{locality} = 13 \times 100\% / 21 = 62\%$). Thus, we claim that:

Claim 2: User activities, that are reused, are attributed to locality. $R_{locality}$ is a better estimate of reuse opportunities than the recurrence rate. In particular, many of the observed UNIX history-tool uses occur in phases and are attributed to history-for-reuse situations.

An important concern in the design of history tools is the development of strategies to predict effectively a small set of user interactions which will be needed in the near future (i.e., history candidates). The working-set policy offers one such history prediction strategy. Greenberg and Witten (1988b) used this strategy to predict reuse candidates and found that a history based on a working set (with $T=10$ submissions) would, on average, contain the desired command about 60% of the time and miss out about 40% of the time. The question is whether the observed performance of the working-set history prediction strategy is near optimal? Our conjecture is that the performance is suboptimal and this can be confirmed by examining the extent of locality in user command references. Recall that computer memory research demonstrated that good locality is an important prerequisite for the optimal performance of the working-set policy. Specifically, we claim that:

Claim 3: Because command line references exhibit poor locality, a working-set history prediction strategy would perform suboptimally.

In order to prove this claim, we compare the relative merits of this strategy when locality is good (i.e., optimal performance of the working-set policy) and when locality is poor (i.e., suboptimal performance of the working-set policy).

The studies, described in the balance of this chapter, examine the phenomenon of locality in command line references in UNIX. Study 1 uses Madison and Batson (1976)'s locality-detection method to determine if locality is indeed present in command line references and if so, to characterize the extent of locality in command line references. We conducted Studies 2 and 3 to corroborate Claims 1-3. Study 2 examines Claim 1 – the question of whether locality is a random behaviour or an artifact of user interactions. Study 3 provides evidence to support Claims 2 and 3 (i.e., locality accounts for history usage and good locality in command line references can enhance the working-set strategy for predicting reuse candidates). Prior to describing these studies, user traces used in the studies are described.

Users' Session Traces

Session traces used in subsequent studies are those collected by Greenberg (1988a). A total of 168 users from four groups participated in his study:

- a) *experienced programmers* — 36 senior computer science undergraduates with a fair knowledge of programming languages and the UNIX environment.
- b) *computer scientists* — 52 faculty, graduates and researchers from the Computer Science Department at the University of Calgary having varying experience with UNIX, but all being experts with computers in general.
- c) *non-programmers* — 25 office staff and members of the Faculty of Environment Design at the University of Calgary who had minimal knowledge of UNIX and only used UNIX applications (e.g., word-processing).
- d) *novice programmers* — 55 students from an introductory Pascal course with little or no previous exposure to programming, operating systems or UNIX-like command-based interfaces.

Each user's trace data was strung together as one long session in which session boundary commands like "logout" were retained. Command lines in which an error occurred were excluded from the session. Each command line had a note indicating whether *cs* history was used to help form the command line but nothing indicating which history feature was used.

Study 1 : Does Locality Exist in User Interactions?

The initial question is: "Does locality, by Madison and Batson (1976)'s definition, exist in user interactions?" This question is explored using two different criteria to qualify repetition of user directives. The first criterion is based on the repetition of full command lines (known as the command-lines case) while the second criterion is based on the repetition of command names only (known as the command-names case). Descriptive statistics were collected to characterize the

extent of the locality (i.e., $R_{locality}$) and the extent of the recurrence (i.e., R) in each user session trace.

Results and Discussion

Locality was observed in both the command-lines case and the command-names case. Table 1 summarizes the locality set sizes that were observed in each case and the number of subjects, in each group and the sample as whole, that produced these locality set sizes. In each of the two subsequent sections, we examine the nature of locality in each case.

Locality in Full Command Lines

The recurrence criterion in the command-lines case is fairly restrictive and makes the algorithm sensitive to any variation in the command lines (i.e., a different ordering of the command line objects would terminate a locality set). Despite this restriction, locality set sizes 1 to 12 and 14 were observed with 135 of the 168 users producing a locality set size of 4, 49 of the 168 users producing a locality set size of 6, 20 of the 168 users producing a locality set size of 8, and a small number of users producing locality set sizes larger than 8 (see Cmd Line columns in Table 1).

Figure 5 provides, by way of locality maps from one subject in each of the four groups, a visual characterization of the nature and extent of locality exhibited by the 168 subjects. The locality maps illustrate that while the extent of locality and the locality set sizes vary, locality is exhibited at various periods in a user session. Table 2 summarizes, by way of descriptive statistics, the extent of recurrence (i.e., R), the extent of locality (i.e., $R_{locality}$), and the percentage of R that was accounted for by $R_{locality}$.

Many of the observed locality set sizes were attributed to novice programmers (i.e., 50, 18, and 8 novice programmers produced locality set sizes 4, 6, and 8, respectively). On average, novice programmers exhibited locality in 51% of their sessions. 62% of their recurrences ($R = 81%$) were accounted for by locality ($R_{locality} = 51%$). The explanation for both of these results is that novice users do not have a large command vocabulary and do not deviate substantially in the commands they issue. In comparison, users in the other three user groups exhibited far less locality (see Table 2) because they possess more skill and are more likely to vary the construction of their command lines.

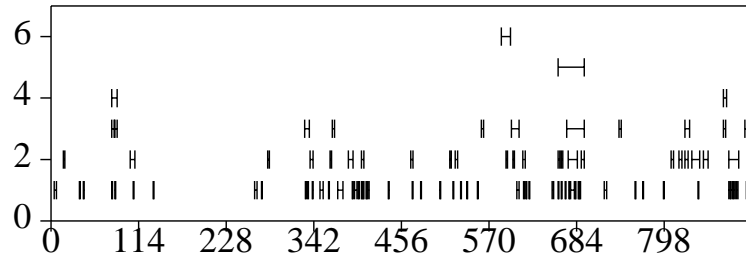
For each user and each observed locality set size, the average number of full command lines in a phase and the average number of occurrences of the same locality set were determined. Then, for each user group and the sample as a whole, the mean and standard error for each of the two measures were computed. Figure 6 plots the mean for the first measure (i.e., average number of command lines in a phase) and Table 3 lists the mean and standard error for the first six locality set sizes. Figure 7 plots the mean for the second measure (i.e., average number of occurrences of the same locality set) and Table 4 lists the mean and standard error for the first six locality set sizes. The graphs and tables illustrate differences across the four groups for each of the two measures.

Set Sizes	Computer Scientists		Experienced Programmers		Non Programmers		Novice Programmers		All Subjects	
	Cmd Line	Cmd Name	Cmd Line	Cmd Name	Cmd Line	Cmd Name	Cmd Line	Cmd Name	Cmd Line	Cmd Name
1	52	52	36	36	25	25	55	55	168	168
2	51	52	36	36	25	25	55	55	167	168
3	47	52	33	36	21	25	54	55	155	168
4	38	50	31	34	16	23	50	55	135	162
5	21	44	24	31	8	20	33	49	86	144
6	9	35	16	25	6	15	18	41	49	116
7	5	21	9	20	2	15	6	33	22	89
8	4	13	7	13	1	7	8	29	20	62
9	1	7	3	10		6		18	4	41
10	1	6	2	7		3		14	3	30
11	1	5	1	3		1		13	2	22
12		1	1			3		6	1	10
13		1		2				4		7
14			2	3				3	2	6
16								1		1
17								1		1
20								1		1
29								1		1

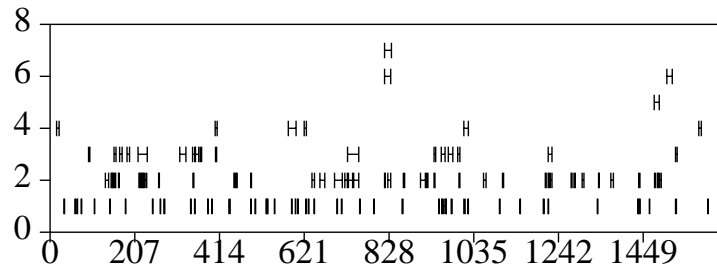
Table 1: The number of users observed producing the locality set sizes for the command-lines and command-names criteria.

User Group	No. of Users	R		$R_{locality}$		% of R Accounted for by $R_{locality}$ $R_{locality} \times 100\% / R$	
		Mean	Std. Err.	Mean	Std. Err.	Mean	Std. Err.
Computer scientists	52	69.4	1.1	17.0	1.2	24.4	1.6
Experienced programmers	36	77.7	2.0	26.7	2.5	32.9	2.3
Non programmers	25	68.3	1.7	25.0	2.8	36.1	3.6
Novice programmers	55	80.5	1.0	50.7	2.0	62.3	2.0
All subjects	168	74.6	.8	31.3	1.5	40.4	1.6

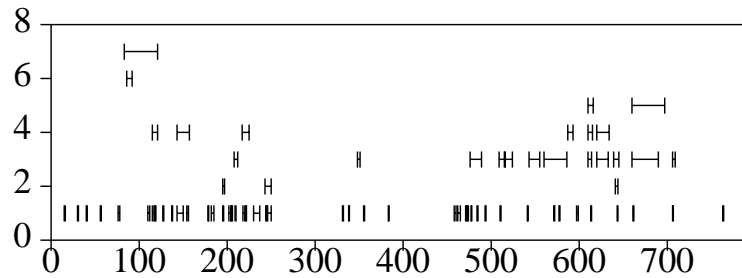
Table 2: For all users and the command-lines criterion, the mean and standard error of R , $R_{locality}$, and percentage of R that was accounted for by $R_{locality}$ are tabulated.



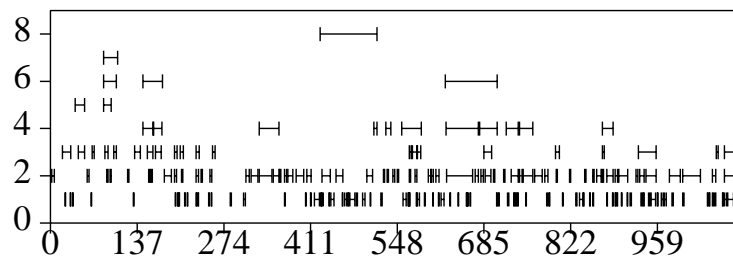
computer-scientist-3 with session length of 909 lines



experienced-programmer-34 with session length of 1651 lines



non-programmer-16 with session length of 795 lines



novice-programmer-15 with session length of 1091 lines

Figure 5: Phase diagrams for four of the users, one from each user group for the command-lines criterion. The bars delimit phases, the y-axis is the locality set sizes, and the x-axis is the command line number in a user session.

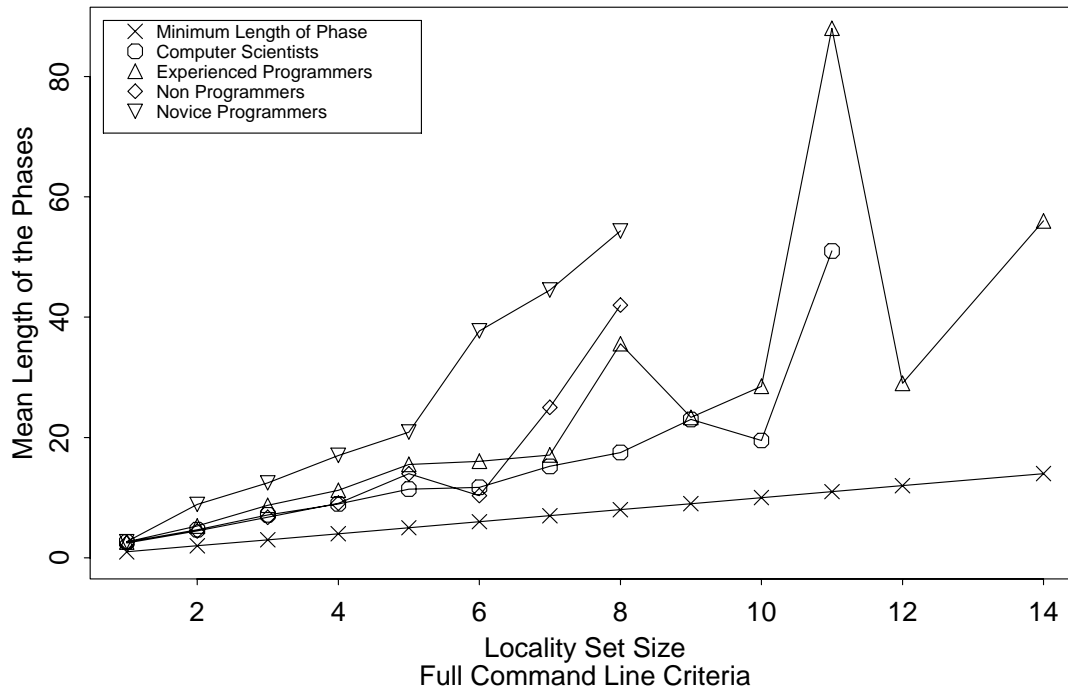


Figure 6: A plot of the mean number of full command lines in a phase for each user group based on user means as a function of locality set size.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	2.62	4.68	7.14	8.96	11.42	11.72
	Std. Err.	0.11	0.27	0.35	0.53	1.34	1.44
Experienced programmers	Mean	2.64	5.28	8.72	11.22	15.52	16.05
	Std. Err.	0.08	0.29	0.61	1.77	1.78	1.76
Non programmers	Mean	2.48	4.47	6.74	9.07	13.97	10.33
	Std. Err.	0.12	0.21	0.40	0.86	1.85	1.09
Novice programmers	Mean	2.68	8.87	12.46	17.01	20.90	37.70
	Std. Err.	0.05	0.52	0.90	1.05	2.31	6.08
All subjects	Mean	2.62	6.16	9.27	12.48	16.44	22.51
	Std. Err.	0.05	0.25	0.41	0.66	1.15	2.84

Table 3: For the command-lines criterion, the mean and standard error (σ/\sqrt{n}) of the number of command lines in a phase for locality set sizes 1 to 6. For instance, the mean number of command lines in a phase for a locality set of size 1 for subjects in the Computer Scientist group is 2.62 command lines.

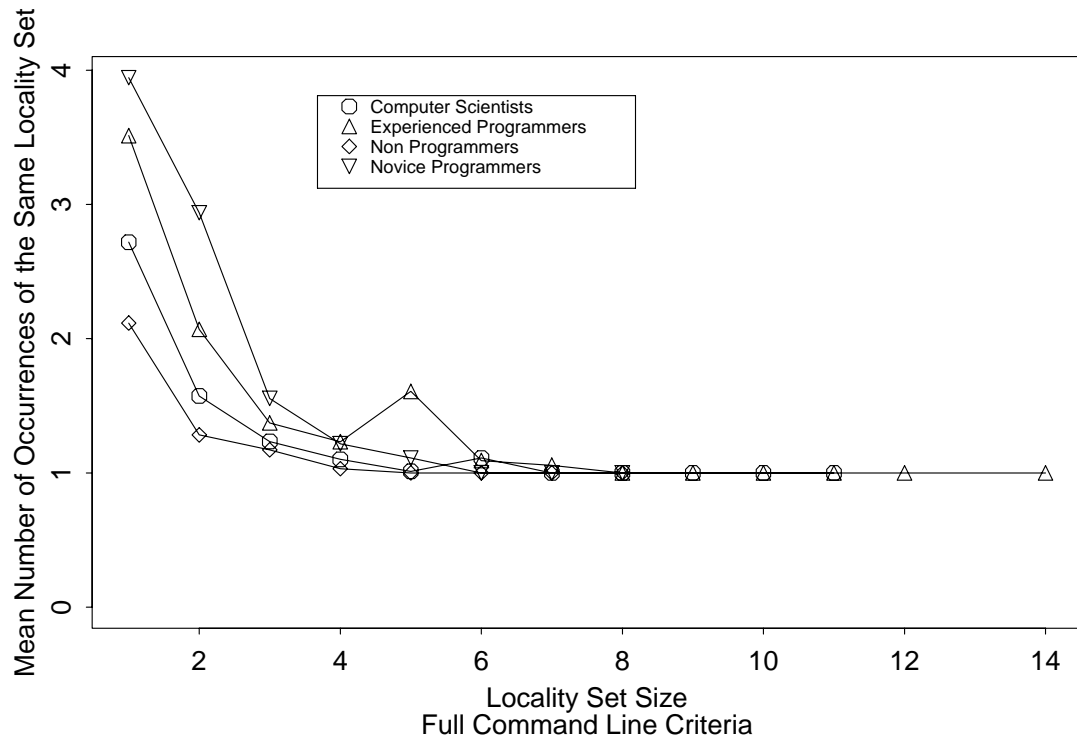


Figure 7: A plot of the mean number of occurrences of the same locality set for each user group based on user means as a function of locality set size for the command-lines criterion.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	2.72	1.57	1.23	1.10	1.01	1.11
	Std. Err.	0.16	0.07	0.06	0.04	0.01	0.11
Experienced programmers	Mean	3.51	2.07	1.37	1.23	1.61	1.09
	Std. Err.	0.38	0.20	0.09	0.10	0.27	0.06
Non programmers	Mean	2.12	1.28	1.17	1.03	1.00	1.00
	Std. Err.	0.19	0.06	0.06	0.03		
Novice programmers	Mean	3.94	2.94	1.56	1.22	1.11	1.00
	Std. Err.	0.32	0.17	0.10	0.06	0.04	
All subjects	Mean	3.20	2.09	1.37	1.17	1.22	1.05
	Std. Err.	0.15	0.09	0.05	0.03	0.08	

Table 4: For the command-lines criterion, the mean and standard error of the number of occurrences of the same locality set for sizes 1 to 6. For instance, the mean number of occurrences of the same locality set of size 1 for subjects in the Computer Scientist group is 2.72.

Several users in the experienced programmer group with extensive programming and UNIX experience generated locality set sizes of 9 and larger. This is not surprising because experienced programmers have larger command sets, perform sophisticated tasks, and have very well-learned behaviours. In contrast, all but one of the subjects in the other three user groups have locality set sizes of 8 and less because these subjects have smaller command sets and less experience with UNIX and *cs.h*.

An examination of the mean number of occurrences of the same locality sets reveals that users tended to repeat the same locality sets for smaller set sizes (especially locality set sizes 1 and 2) more than for larger set sizes (i.e., size 4 and up). A comparison of user groups with respect to the mean number of occurrences of the same locality set indicates that novice and experienced programmers repeat the same locality set more often (see Table 4 and Figure 7).

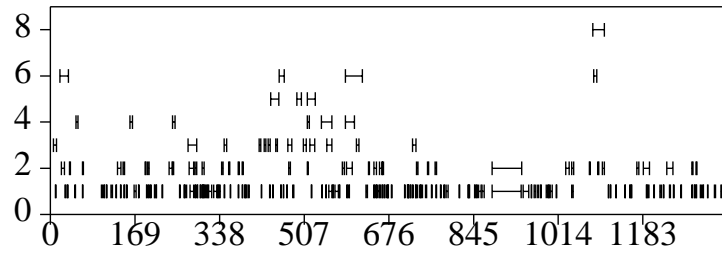
Locality in Command Names Only

Locality set sizes 1 .. 14, 16, 17, 20, and 29 were observed in the command-names case (see Table 1). Figures 8, 9, and 10 and Tables 1, 5, 6, and 7 summarize the extent of locality observed in the command-names case. Larger locality sets were observed because activities repeat if command names repeat.

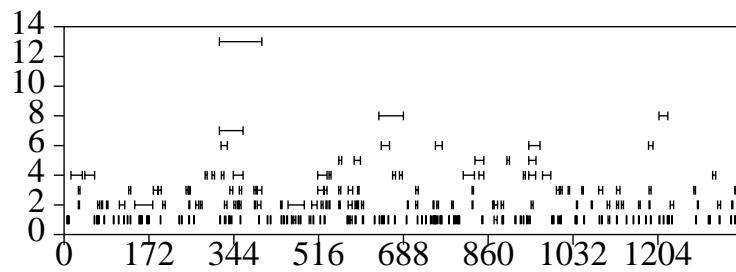
Many of the larger locality set sizes were produced by novice programmers. On average, novice programmers have longer phases and repeated the same locality set more often than the other three user groups. Unlike the command-lines case, non-programmers, on average, have the next longest phases and they repeated the same locality set about as often as novice programmers. In general, computer scientists and experienced programmers were comparable in terms of the size of their locality sets, the average number of command names in their phases, and the average number of occurrences of the same locality set.

User Group	No. of Users	R		$R_{locality}$		% of R Accounted for by $R_{locality}$ $R_{locality} \times 100\% / R$	
		Mean	Std. Err.	Mean	Std. Err.	Mean	Std. Err.
Computer scientists	52	96.0	.4	52.3	1.8	55.0	1.9
Experienced programmers	36	95.2	.5	53.9	2.0	56.6	2.1
Non programmers	25	95.2	.7	75.4	3.3	79.1	3.3
Novice programmers	55	97.4	.2	82.9	1.6	85.1	1.6
All subjects	168	96.0	.2	66.1	1.5	68.8	1.5

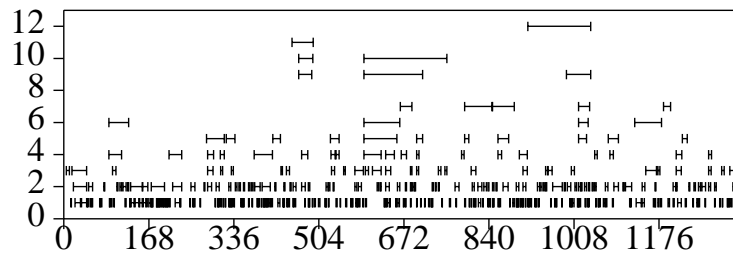
Table 5: For command-names criterion, the mean and standard error of R , $R_{locality}$, and percentage of R that was accounted for by $R_{locality}$.



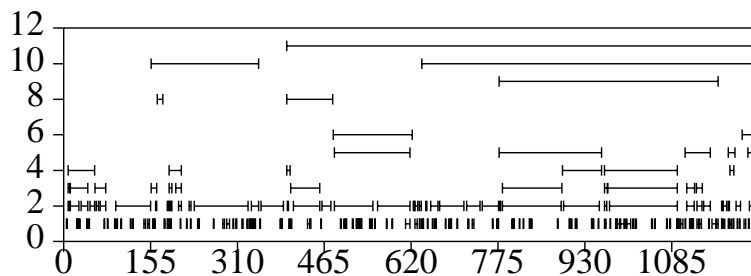
computer-scientist-15 with session length of 1348 lines



experienced-programmer-11 with session length of 1370 lines



non-programmer-18 with session length of 1339 lines



novice-programmer-44 with session length of 1237 lines

Figure 8: Phase diagrams for four of the users, one from each user group for the command-names criterion. The bars delimit phases, the y-axis is the locality set sizes, and the x-axis is the command line number for a user session.

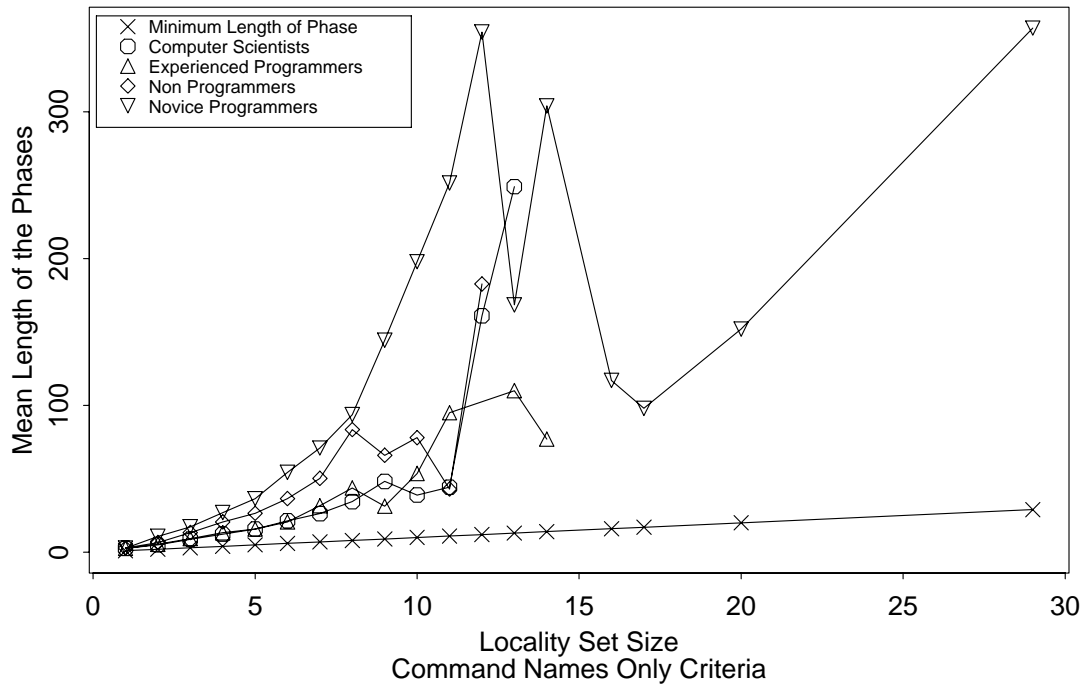


Figure 9: A plot of the mean number of command names in a phase for each user group based on user means as a function of locality set size.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	2.59	5.01	8.82	12.29	15.77	21.39
	Std. Err.	0.05	0.14	0.44	0.75	0.89	1.32
Experienced programmers	Mean	2.64	5.40	9.24	13.24	15.72	20.63
	Std. Err.	0.06	0.19	0.54	1.04	0.84	1.66
Non programmers	Mean	2.84	6.54	13.37	20.59	26.47	36.50
	Std. Err.	0.12	0.39	1.55	3.78	3.74	5.67
Novice programmers	Mean	2.78	10.77	17.31	26.98	36.45	54.37
	Std. Err.	0.04	0.74	1.13	3.40	3.70	6.44
All subjects	Mean	2.70	7.21	12.37	18.66	24.28	34.84
	Std. Err.	0.03	0.32	0.55	1.40	1.60	2.81

Table 6: For command-names criterion, the mean and standard error of the length of phases for locality set sizes 1 to 6. For instance, the mean number of command names in a phase for a locality set of size 1 for subjects in the Computer Scientist group is 2.59 command names.

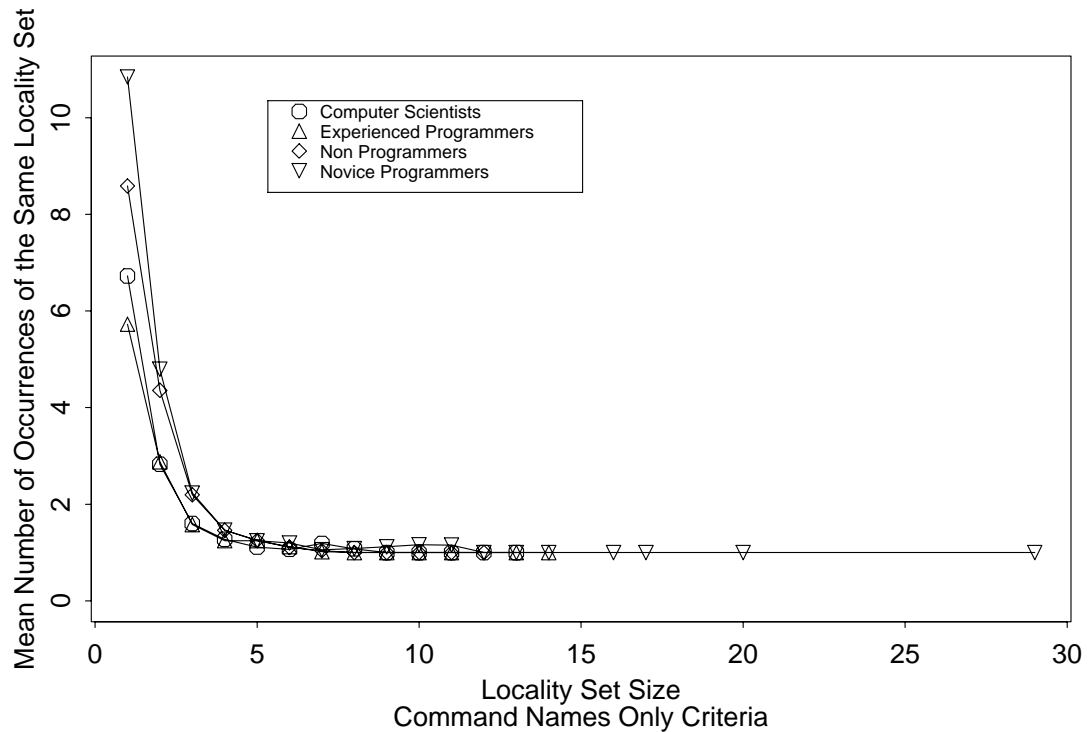


Figure 10: A plot of the mean number of occurrences of the same locality set of for each user group based on user means as a function of locality set size for the command-names criterion.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	6.73	2.83	1.60	1.27	1.11	1.07
	Std. Err.	0.57	0.17	0.07	0.07	0.05	0.03
Experienced programmers	Mean	5.73	2.88	1.58	1.25	1.24	1.12
	Std. Err.	0.44	0.20	0.10	0.05	0.13	0.04
Non programmers	Mean	8.59	4.36	2.19	1.46	1.25	1.11
	Std. Err.	1.31	0.53	0.20	0.13	0.08	0.06
Novice programmers	Mean	10.84	4.79	2.23	1.47	1.24	1.20
	Std. Err.	0.77	0.31	0.19	0.11	0.06	0.06
All subjects	Mean	8.14	3.71	1.89	1.36	1.21	1.13
	Std. Err.	0.41	0.16	0.08	0.05	0.04	0.03

Table 7: For the command-names criterion, the mean and standard error of the number of occurrences of the same locality set for sizes 1 to 6. For instance, the mean number of occurrences of the same locality set of size 1 for subjects in the Computer Scientist group is 6.73.

Study 2 : Does Locality Exist By Chance?

Study 2 answers the question: “Is locality a randomly occurring behaviour or is it a behaviour that arises from user-computer interactions?” Specifically, “Do *pseudo users* generate locality sets in the same proportions as *real users*?” Pseudo users generate randomly sequenced, but syntactically correct, command lines. Note, we answer this question by examining the locality behaviour for the full-command-lines case although examining locality behaviour for the command-names-only case would have been appropriate as well.

Statistical Test

Real-user sessions represent one independent sample drawn from a population. Pseudo-user sessions represent another independent sample drawn from a population. The question is: “Are the locality behaviours exhibited in real and pseudo-user samples those behaviours exhibited by individuals that come from the *same* population?” If so, then both sample means would estimate the same population mean for the locality measure. Otherwise, the sample means estimate two distinct population means. The locality measure $L_l(u)$ for a user u is the proportion of the total observed locality that is of a given locality set size:

$$L_l(u) = \frac{n_l(u)}{N(u)} \times 100\% \quad l = 1 \dots 12, 14 \text{ and } u = p, r.$$

where $n_l(u)$ represents the number of locality sets that were observed for locality set size l for user u 's session, and $N(u) = n_1(u) + \dots + n_{12}(u) + n_{14}(u)$. r denotes a *real-user* sample and p denotes a *pseudo-user* sample. There are 13 such locality measures which correspond to the 13 locality sets observed for real users.

A sampling experiment can be conducted which draws a sample of 168 pseudo-user session traces from the pseudo-user population. For each locality set size l , the 13 sample means for the locality measure $\bar{L}_l(u=p)$ can be measured. This metric estimates the population mean μ for the locality measure $L_l(u)$. When this sampling experiment is repeated a large number of times (i.e., 1000), 13 relative frequency histograms based on the sample means $\bar{L}_l(u=p)$ can be generated, one for each locality set size l . Each relative frequency histogram approximates the probability distribution of the sample mean $\bar{L}_l(u=p)$. Specifically, if another independent sample is drawn from the pseudo-user population, the mean of the locality measures $\bar{L}_l(u=p)$ for this sample would be well described by its relative frequency histogram.

To test the claim that the mean locality measure $\bar{L}_l(u=r)$ for real-user samples comes from the same sampling distribution as $\bar{L}_l(u=p)$, we determined the probability that the mean locality measure of $\bar{L}_l(u=p)$ is equal to or more extreme than $\bar{L}_l(u=r)$. The probability condition is a statistical test of one tail of the sampling distribution. The particular probability condition depends on which tail of the sampling distribution $\bar{L}_l(u=r)$ is located in. A small probability (i.e., $P[\bar{L}_l(u=p) \geq \bar{L}_l(u=r)] < \alpha$ or $P[\bar{L}_l(u=p) \leq \bar{L}_l(u=r)] < \alpha$ where $\alpha = .01$) provides strong evidence that such an event is unlikely to occur if the claim was true. 13 such one-tailed tests were performed, one for each observed locality set l , using the corresponding sampling distribution.

Method

Each sampling distribution for each observed locality set size was obtained from a computer simulation of the sampling experiments. Each sample had 168 sessions corresponding to the 168 sessions in the real-user sample. In the sampling experiment, the sequence of command lines for each real-user session was randomly permuted using a different random number, to form the corresponding pseudo-user session. 1000 pseudo-user samples were generated.

Each session (*real* and *pseudo*) was analyzed by the locality-detection algorithm and a summary of detected locality sets was collected. Each summary included 13 $n_l(u)$ values ($l = 1 \dots 12, 14$) and total number of locality sets observed (i.e., $N(u) = n_1(u) + \dots + n_{12}(u) + n_{14}(u)$). Then, the locality measure, $L_l(u)$, for each user sample u and each set size l was computed.

Sample means for each of the 13 locality measures, $\bar{L}_l(u)$, for a *real-user* sample and 1000 *pseudo-user* samples were determined. Then, for each locality set size l , a frequency histogram for the 1000 *pseudo-user* sample means $\bar{L}_l(u=p)$ was produced. A relative frequency histogram was generated by taking the frequencies and dividing by the total number of samples (i.e., 1000).

Results and Discussion

Figure 11 displays 4 of the 13 sampling distributions of $\bar{L}_l(u=p)$ that were obtained from the sampling experiments. Table 8 lists, for each of the 13 locality set sizes, $\bar{L}_l(u=r)$ values for *real-user* sessions, probability tests, and probability test values P . P values for all locality set sizes were less than .01. The claim that the two groups come from the same population was rejected at the $\alpha = .01$ level. Therefore, locality is a behavioural artifact of user interactions and does not occur by pure chance. Figure 12 illustrates the difference between $\bar{L}_l(u=p)$ and $\bar{L}_l(u=r)$.

Study 3 : Does Locality Enhance the Recurrence Picture?

This is a two-part study which examines two claims concerning two aspects of Greenberg and Witten (1988b)'s study. In both parts of the study, we examine the locality behaviour for the full-command-lines case although it would have been appropriate to examine the locality behaviour for the command-names-only case. First, does locality account for Greenberg and Witten (1988b)'s observed history usage? Second, does limiting the history prediction strategy to locality situations enhance the quality of the prediction?

Study 3A: History-for-Reuse Occurs Within a Locality

Claim 2 states that the fraction of the recurrent activities, those appearing in a locality (i.e., $R_{locality}$), represent a more meaningful estimate of reuse opportunities than the recurrence rate estimate (i.e., R). Recall, R is the percentage of the session activities containing recurrent activities while $R_{locality}$ is the percentage of the session activities containing phase activities.

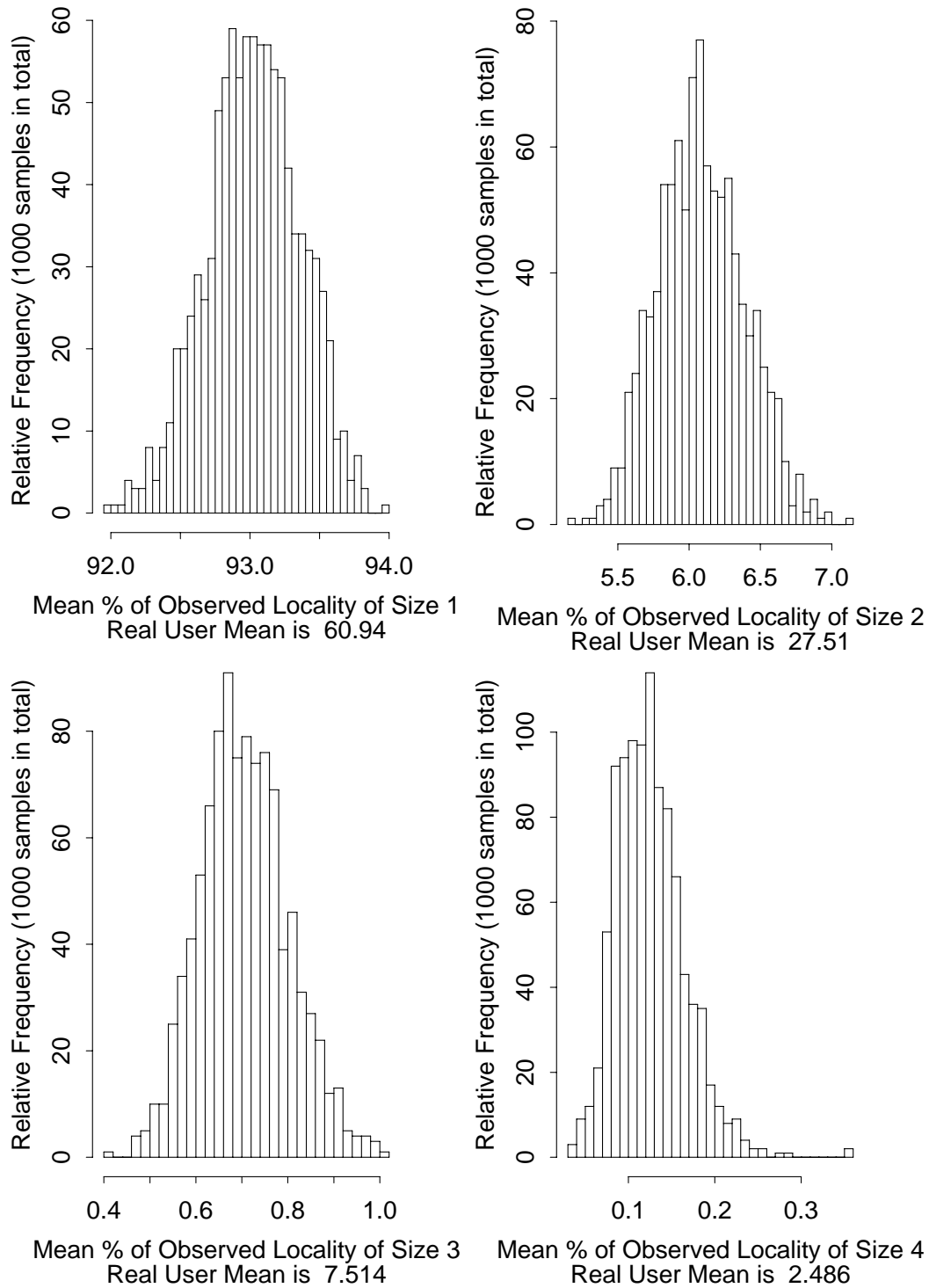


Figure 11: Relative frequency distributions of $\bar{L}_l(u=p)$ for $l = 1 - 4$.

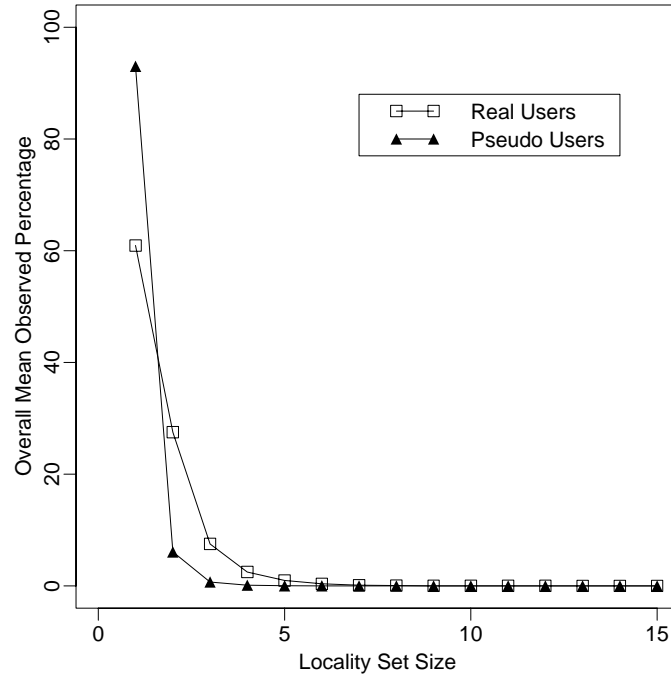


Figure 12: $\bar{L}_l(u=p)$, $\bar{L}_l(u=r)$ versus locality set size.

Size l	$\bar{L}_l(u=r)$	Probability Test	Value
1	60.937	$P[\bar{L}_1(u=p) \leq 60.94]$	0.
2	27.515	$P[\bar{L}_2(u=p) \geq 27.52]$	0.
3	7.514	$P[\bar{L}_3(u=p) \geq 7.51]$	0.
4	2.486	$P[\bar{L}_4(u=p) \geq 2.49]$	0.
5	0.965	$P[\bar{L}_5(u=p) \geq 0.97]$	0.
6	0.368	$P[\bar{L}_6(u=p) \geq 0.37]$	0.
7	0.117	$P[\bar{L}_7(u=p) \geq 0.12]$	0.
8	0.065	$P[\bar{L}_8(u=p) \geq 0.07]$	0.
9	0.009	$P[\bar{L}_9(u=p) \geq 0.01]$	0.004
10	0.008	$P[\bar{L}_{10}(u=p) \geq 0.01]$	0.002
11	0.004	$P[\bar{L}_{11}(u=p) \geq 0]$	0.004
12	0.008	$P[\bar{L}_{12}(u=p) \geq 0.01]$	0.001
14	0.002	$P[\bar{L}_{14}(u=p) \geq 0]$	0.

Table 8: The $\bar{L}_{r,l}$ values and the probability tests for the respective locality set sizes and their corresponding values. The test is the probability that the mean locality measure of $\bar{L}_l(u=p)$ is equal to or more extreme than $\bar{L}_l(u=r)$. It involves the tail of the sampling distribution on which the value of $\bar{L}_l(u=r)$ is located (see Figure 11 for the location of the value of $\bar{L}_l(u=r)$ in the sampling distribution for $l=1, 2, 3, 4$).

Method

In order to verify this claim, we conduct two selection experiments. In the *locality-selection* experiment, we identify those users who made use of the history tool. Then, for these user's session, we select only those activities appearing in phases and count the number of activities that are formed using a history tool (i.e., $h_{locality}$). This selection experiment would select a percentage of the activities in a user's session that is equal to $R_{locality}$. In the *random-selection* experiment, we randomly select the same fraction of the activities in the same user's sessions and count the number of activities that are formed using a history tool (i.e., h_{random}). Then, we determine $H_{locality}$ and H_{random} . $H_{locality}$ is the percentage of the activities appearing in phases that are formed with a history tool:

$$H_{locality} = \frac{h_{locality}}{h_{total}} \times 100\%$$

H_{random} is the percentage of the randomly selected activities that are formed using a history tool:

$$H_{random} = \frac{h_{random}}{h_{total}} \times 100\%$$

Note, the value of H_{random} is equal to $R_{locality}$ because a random selection of a percentage of the activities in a user's session (i.e., $R_{locality}$) would also randomly select a similar percentage of the activities formed using a history tool. Thus, the value of h_{random} is:

$$h_{random} = \frac{R_{locality} \times h_{total}}{100}$$

If Claim 2 is false, then activities formed using history tools would be equally likely in phases as well as transitions. That is, the percentage of activities in a user session formed with a UNIX history tool within phases $H_{locality}$ would be equal to H_{random} and $R_{locality}$.

However, if Claim 2 is true then the value of $H_{locality}$ would be substantially greater than the value of H_{random} (i.e., $H_{locality}$ is greater than H_{random} which is equal to $R_{locality}$). Thus, $h_{locality}$ should be substantially larger than h_{random} . Note that only 25% of the session activities (i.e., $R_{locality} = 25\%$ which is based on user sessions that involved history use) occur in a phase and less than 4% of the session activities involve history use; both are small percentages (see Table 9). Thus, a valid claim means that the selection systematically locates a large percentage of history use in a quarter of the command lines.

We conducted the locality-selection experiment on the sessions for those users who used the UNIX history tool. For each of these users, we determined the values of $h_{locality}$, h_{total} , h_{random} and $H_{locality}$.

Results and Discussion

A within-subjects ANOVA test comparing h_{random} and $h_{locality}$ yields a significant difference ($F(1,88) = 65.1$, $p = 0$). This confirms our claim that the extent of locality $R_{locality}$ is a better estimator of reuse than the recurrence rate R . On average, 65% of the command lines involving history use are issued within a

User Group	No. of Users	$R_{locality}$		% of the History Usage Occurring in a Locality (H)		% of the Total Session Involving History Usage	
		Mean	Std. Err.	Mean	Std. Err.	Mean	Std. Err.
Computer scientists	37	16.3	1.5	60.7	3.8	4.1	.7
Experienced programmers	32	26.5	2.6	64.4	3.7	4.5	.6
Non programmers	9	25.7	5.9	60.4	12.1	4.4	2.1
Novice programmers	11	46.7	5.3	83.1	6.1	2.1	.9
All subjects	89	24.7	1.7	64.8	2.6	4.1	.4

Table 9: The mean and standard error of $R_{locality}$, H and % of total sessions involving history usage. Unlike the $R_{locality}$ values in Table 2, these values are computed from only session traces of history users.

locality (see Table 9). This figure is two and a half times the number of command lines involving history use that are selected by a systematic process (i.e., locality) compared to that which a random process would have selected.

However, on average, 35% of history uses³ are not found inside a phase. One possible explanation for the unaccounted 35% is that history uses may not be limited to the literal recall of a history item; they may include the modification and recall of parts of previous command lines. If such history features (i.e., modification and partial recall) are used to form a command line, the modified command line would break the locality set. Thus, the locality detection method must be refined to alleviate the restricted definition of recurrences (i.e., literal repetition of previous command lines) so that the repetition of modified command lines is included in a locality set. It is difficult to verify directly our explanation for the missing 35% history usage because our user session traces are not annotated with the history feature that a user used. However, this explanation is indirectly supported by two findings from our studies:

- (1) In our exploratory study, we found that *cs/h* users used word designators and modifiers (see Table 3 in Chapter 4). These features permit the recall and modification of parts of previous command lines.
- (2) In this study, we note that 83% of the novice programmers' history uses were accounted for by locality. This group of subjects were students taking an introductory programming course with no previous exposure to UNIX and *cs/h* history. Thus, their use of history would be limited largely to the simple recall of previous command lines.

³ Recall, the locality-detection method requires that the composition and makeup of individual items of a locality set remain unchanged throughout the phase. Thus, the 65% of history uses found in phases are the history uses involving the literal reuse of commands.

Study 3B: Locality and Prediction of Reuse Candidates

To corroborate the claim that the performance of the working-set history prediction strategy (observed by Greenberg and Witten (1988b)) is suboptimal, one needs to demonstrate two things. First, user sessions must exhibit poor locality. Second, the performance of the working-set strategy during locality periods is shown to be better than during the entire period of a user's session.

Method

Two versions of each user's session are required: a *full user session* S and a *restricted user session* S' . S' contains only command lines in S that are part of phases or phase formations. Each S and S' trace is run through the working-set algorithm, outlined in Figure 13, using various window sizes, $T = 1 .. 10, 20, 30, 40, 50$. For each user and each T , the percentage of time that the next command line is located within the current working set is computed. Also, for each T , the average of this percentage for all users is computed.

Results and Discussion

As revealed in the full-command-lines case, on average, the extent of locality $R_{locality}$ over a full user session S is 31% (see Table 10). This measure is substantially less than $R = 74\%$. However, $R_{locality}$ for the locality-only portions of a user session S' is 90% and R is 88%. By removing command lines appearing in transitions, the recurrence rate R and the extent of the locality $R_{locality}$ are much higher. The resulting traces exhibit good locality and high recurrence rate, and more importantly, the two metrics are essentially equivalent.

Figure 14 shows the performance of the prediction strategy for various working-set window sizes when prediction is restricted to locality periods only and when it is performed throughout a session. Hit percentages for the two predictions are tabulated in Table 11. These results corroborate the computer memory research finding which found that when locality is good (see S'), so is the performance of the working-set history prediction strategy. Thus, the performance of the working-set history prediction strategy (S) is suboptimal⁴.

With the restricted user sessions, reuse prediction is, on average, 92% for S' compared to 67% for S with a window size of $T = 10$. In general, hit percentages for a restricted user session (S') are higher than for a full session S at the same recurrence distance T (i.e., $1 \leq T \leq 50$). For $T=10$, the differential in prediction performance is 26%. Doubling T results in a smaller differential in prediction performance (18%) compared to that obtained for $T=10$. The incremental gain for $T > 3$ of restricted user sessions diminishes in comparison to full sessions. From the data in Table 11, $T=3$ appears to be the threshold point.

⁴ Another finding of computer memory research is that the working-set policy is the most likely, among nonlookahead policies to generate minimum space-time for any given program. In fact, Denning (1980) reports that experiments with real programs have found that the working-set policy can be run with a single global control parameter value (i.e., T) and deliver performance typically no worse than 10 percent from optimum.

Given:

- an array *cmd_line* which holds *n* command lines in a session
- an array *num_hits* to accumulate the number of hits at various distances
- an array *T* to hold the *T* values of interest
- an array *hit_percent* to store the cumulative hit % for a particular *T*

```
int T[14] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50};

/* scan each item to find its nearest preceding match and note statistics */
for (i = 1 to n)
    for (j = i to 1 by -1)
        if (cmd_line[i] == cmd_line[j]) then {
            distance = i-j
            no_hits[distance] = distance + 1
            break;
        }

/* compute the hit_percent for the distances indicated in T */
total = 0
t = 0
for (distance = 1 to n) {
    total = total + no_hits[distance]
    if (distance == T[t]) {
        hit_percent[t] = (total/n) * 100
        t = t + 1
    }
}
```

Figure 13: A working set algorithm, from Greenberg (1988b), for determining hit percentage for window sizes $T = 1 .. 10, 20, 30, 40, 50$.

User Group	Statistic	R	$R_{locality}$	R'	$R'_{locality}$
Computer scientists	Mean	69.4	17.0	85.7	89.5
	Std. Err.	1.1	1.2	1.0	.6
Experienced programmers	Mean	77.7	26.7	89.2	88.4
	Std. Err.	2.0	2.5	1.3	.6
Non programmers	Mean	68.3	25.0	80.9	87.5
	Std. Err.	1.7	2.8	1.8	.9
Novice programmers	Mean	80.5	50.7	93.1	91.5
	Std. Err.	1.0	2.0	.5	.4
All subjects	Mean	74.6	31.3	88.1	89.6
	Std. Err.	.8	1.5	.6	.3

Table 10: The recurrence and locality metric values for the full user sessions S and the restricted user sessions S' for each user group and the sample as a whole.

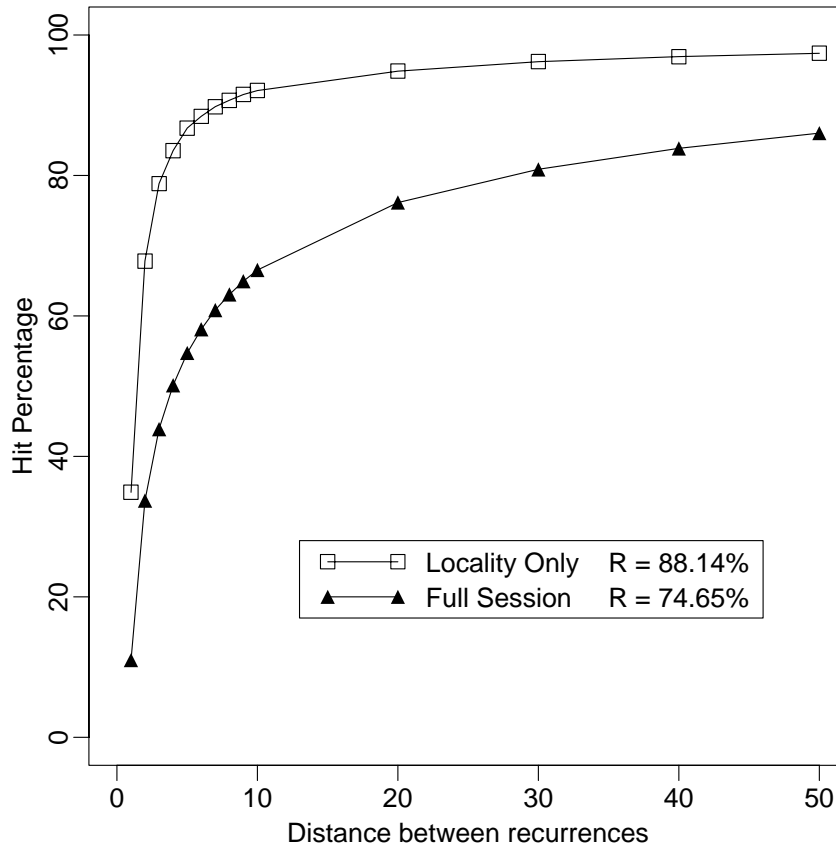


Figure 14: Hit percentage as a function of distance between recurrence.

T	S'	S	$S' - S$
1	34.9	11.0	23.9
2	67.8	33.7	34.1
3	78.8	43.9	34.9
4	83.5	50.1	33.4
5	86.7	54.7	32.0
6	88.4	58.1	30.3
7	89.8	60.8	29.0
8	90.7	63.1	27.6
9	91.5	64.9	26.6
10	92.1	66.5	25.6
20	94.9	76.2	18.7
30	96.2	80.9	15.3
40	96.9	83.9	13.0
50	97.4	86.0	11.4

Table 11: Hit % for various values of T for a full user session (S), restricted user session (S'), and $S' - S$.

These results also demonstrate that a high recurrence rate in command line references is not sufficient for a working-set policy to perform optimally. Rather, the crucial factor is goodness of locality exhibited by command line references. This is consistent with the computer memory research findings [Denning, 1980].

Since users do not exhibit good locality, the working-set history prediction strategy will perform suboptimally. Conditioning techniques, like those proposed by Greenberg and Witten (1988b), may help enhance the prediction of useful history candidates. However, as the above performance analysis clearly demonstrates, history prediction during locality periods are more accurate than during the whole session. Furthermore, the results in Study 3A indicate that 65% of history uses occur within phases.

Concluding Remarks

There are four key results from these studies of locality in command line references. First, locality was exhibited in both the command-lines and command-names cases by all 168 subjects who represent a diverse cross section of UNIX users. Second, locality is a natural consequence of user interactions and is not a randomly occurring behaviour. Third, a substantial percentage of history usages (i.e., 65%) occurred within locality periods. Fourth, unlike program memory references which exhibit good locality (i.e., > 90%), command line references exhibit poor locality (i.e., 31%). This finding has implications for characterizing reuse opportunities and reuse candidates because locality takes into account logical clusters of command lines.

Our study of locality in user interactions along with Greenberg and Witten (1988b)'s study of recency in user interactions, provide an extensive examination of the applicability of computer memory research to the study of user command line reference behaviour and to the performance of the working-set strategy for predicting the (small) set of command line references that a user may need in the near future. However, further research into other aspects of user behaviour are needed before one can conclude that the concepts – working set and locality – are applicable, in general, to other situations in human-computer interaction.

In addition, further refinements to the locality-detection algorithm are required to allow for less restrictive determinations of a locality. The locality-detection technique used in the analyses requires that the references remain unchanged for the duration of a phase. As a result, the pattern matching component is sensitive to slight changes or re-ordering of objects that make up a user reference. The essence of locality is captured by this definition of locality, but in an overly restrictive manner.

An alternative approach for identifying locality, described in Denning (1980), is based on the frequency of segment faults (i.e., transitions are heralded by a series of segment faults occurring in close succession). An examination of this approach of identifying locality, along with a comparison of the results obtained from using this technique against results obtained in our studies, would be informative. This study is left for future research.

In conclusion, there are two possible accounts for poor locality in user interactions. First, poor locality may be due to the fact that users switch between

parallel activities. By separating the different threads of parallel activities, phases would be longer. In this case, the poor locality finding would suggest that user interfaces need to provide tools and resources for managing parallel activities and thereby support for sustaining locality for longer periods. Second, poor locality may be due to the fact that a system imposes fewer constraints on users. As a result, users are not forced to use the same method to accomplish their tasks; they can use different variations which would result in less locality in user interactions. Both accounts are conjectures and must be substantiated through a formal investigation into the cause of poor locality.



Chapter 6

User Effort in the Command Specification Task

The task-artifact and behavioural studies, described in Chapters 2-5, examine two different perspectives of the general question of why users do not make greater use of history tools. Those studies have shown two possible reasons: integrated history tools do not exist to support the seven uses of history tools and designs of history-for-reuse tools do not take locality of user recurrences into account. This chapter explores a third perspective related to the user effort involved in issuing a recurrent operation. In particular, we explore the question: “Does the effort involved in using history tools to specify a recurrent operation outweigh the effort involved in simply specifying the operation from scratch?”

Also, our study uses a cognitive modelling framework to generate models describing the human information processing operations and performance times for the typing and history-tool methods. Since these models represent the cornerstone of our analysis of user effort, it is important that their predictions be reasonably accurate. Therefore, we also examine the error in the models’ predictions.

We begin with a description of our user effort study and its findings. Then, we present an experiment we conducted to compare the predicted times against the observed times. Finally, we make some remarks regarding the cognitive modelling methodology and ideas for future work.

User Effort Study

The primary objective of this study is to compare and contrast the user effort associated with various ways of specifying recurrent commands based on an analysis of the human information processing requirements. The next few subsections describe the following aspects of the study.

- analytic methodology
- user effort
- command-specification task and methods
- constraints of the study
- estimates for predictions
- procedure
- results and discussion

Analytic Methodology

Cognitive modelling is a description and analysis of the knowledge and sub-tasks used to carry out a task. This type of a *task analysis* is carried out with the benefit of psychological models of human behaviour. The knowledge and task descriptions produced in the analysis are cognitive models. These models provide predictions about user behaviour (e.g., consistency, performance times).

There are four reasons for using cognitive modelling. First, it employs psychological theories of human behaviour. Hence, predictions and evaluations are grounded on these theories. Second, a goal of cognitive modelling is to strive to predict average user behaviour to within 80% accuracy while expending 20% of the effort normally required in prototyping and testing [John, 1988]. Thus, cognitive models provide designers with analytic tools to predict, prior to prototyping, how users will interact with computer systems in an approximate and reasonably accurate manner. Third, cognitive modelling provides a scientific foundation upon which we can systematically investigate and extend our investigations of user effort. Specifically, we can extend our current study within this framework to include tasks involving partial reuse and modification of recurrent commands or even command generation.

Our study uses a cognitive modelling framework consisting of the Model Human Processor (MHP for short) and Goals, Operations, Methods, and Selection Rules (GOMS, for short). MHP is a general characterization of human information processing. It consists of a system architecture and a set of quantitative parameters of component performance. The architecture is made up of three processors – perceptual, cognitive, and motor – and their associated memories, and a set of principles of operation. Each processor operates serially within itself and concurrently with all the others, subject to serial limitations imposed by data flow. Elemental acts of the processors are referred to as *operators*.

GOMS is a representation for describing the user knowledge and actions needed to perform a task. A task analysis in this framework involves subdividing a task into the gross functions performed by each MHP processor and decomposing these functions into sequences of operators. The resulting cognitive model is a description of the sequence of human information processing operators involved in carrying out a user task. One of the principal predictions made by these cognitive models is task performance times. MHP assumes that operators act independently of each other so that the time to perform a sequence of operators is the sum of the times to perform each operator.

The MHP/GOMS framework permits the formulation of GOMS models at different levels of task analysis (e.g., unit task, keystroke, and immediate behaviour

levels). The Keystroke Level Model is a GOMS model formulated at the level of the MHP processes. It predicts the task performance time given the sequence of operations for performing a task. The GOMS Model of Immediate Behaviour is formulated within the MHP architecture [John, 1988]. Immediate behaviour is the direct mapping of a stimulus into a response without problem solving or planning. Such behaviours are routine and are generally associated with reaction-time tasks where the response to a stimulus is well known and initiated upon presentation of the stimulus without deliberation about what is appropriate (i.e., stimulus-response compatibility tasks – SRC for short). Our study concentrates on command specification which includes many of the routine activities bridged by the keystroke and immediate behaviour levels. Hence, our study makes use of both GOMS models.

A longer description of the MHP/GOMS framework, the task description language and the cost prediction scheme appears in Appendix C. An in-depth description of this framework, its application, and its various formulations can be found in Card, Moran, and Newell (1983) and John (1988).

User Effort

There are two user effort components: *mental* and *physical* effort. Mental effort is expended when cognitive and visual processing resources are used while physical effort is expended when motor processing resources are used.

We chose to examine user effort for three reasons. First, user effort is a factor which can influence a user's decision to use history tools. Second, we suggested in our studies of user recurrences that history tools can alleviate the physical effort in issuing a recurrent command, and a formal study of user effort would corroborate this suggestion. Third, unlike many other factors influencing history use, which are rather subjective or qualitative, user effort can be objectively examined and quantified.

Since we use the MHP/GOMS framework in our study, the user effort predictions are functions of the types and numbers of elemental MHP acts (i.e., operators) and the task performance times. The numbers and types of MHP operators are derived from a task analysis of each proposed method of command specification. Task performance time is measured from the time after users generate commands in their mind to the time they submit them to a system.

Command-Specification Task and Methods

There are three subtasks involved in issuing a command: *command generation*, *method choice*, and *command specification*. Command generation is the task of mentally generating the command to be issued. Method choice is the task of choosing the method to externalize the desired command. Command specification is the task of issuing the desired command. Our user effort analysis focuses on the command-specification task. In particular, we are interested in the user effort associated with issuing the desired command for a given method and design.

Our analysis does not deal with method choice because this decision-making task is driven primarily by subjective considerations (e.g., personal preferences,

methods that come immediately to mind, user's awareness of the available methods) and because cognitive models do not address such considerations. Therefore, we opted to study user effort in an objective manner (i.e., given a method and design, determine the user effort).

There are four reasons for ignoring a command-generation task. First, command generation involves plan generation and problem solving, which are beyond the scope of current cognitive modelling efforts. Second, the GOMS models used in our analysis assume that user behaviours are routine and involve no planning or problem solving. Third, command generation is poorly understood in HCI (i.e., is command specification intertwined with or separate from command generation?). Finally, a study of command generation constitutes several theses in itself.

Because our understanding of the command generation task is limited, we made two simplifying assumptions for our study. First, we assume that command generation and command specification are independent. Second, we assume that users have generated the full command prior to issuing it and this command is available in working memory. These assumptions permit an initial investigation of the issue of user effort involved in issuing a command. However, they may be a gross simplification of what happens typically when users issue commands. Hence, these assumptions represent a limitation of our study.

Two methods are available for specifying a recurrent command: typing and using a history tool. Three history-tool designs involving recall and/or recognition capabilities are examined. In the *command feature* class, a user *recalls* a feature of the desired command to be invoked; in the *history menu* class, a user *recognizes* the desired command from a visual display of the history; and in the *step buffer* class, a user uses one or a combination of recall and recognition capabilities to peruse and select individual history items.

Command Feature

This class of designs supports the recall of a command from a user history based on a feature that the user provides to uniquely identify it. The four proposed designs are based on different capabilities supported by *csh*.

Name – *!name* repeats the most recent command with the given *name*.

Number – *!number* repeats the command with the given *number*.

Argument – *!?argument* repeats the most recent command with the *argument*.

Last Command – *!!* repeats the most recent command.

History Menu

This class of designs presents a portion of a user history in a menu whose items are selected using a mouse. Two variations are considered:

Popup – A user brings up the menu and scans and selects the desired command.

Static – A user scans a static menu and selects the desired command.

Step Buffer

This class of designs allows the user to single step through a user history from the most recent to the least recent to locate the desired command. Two of the designs are directly supported in *tcs*h while the third is based on a related feature available in *c*sh and *tcs*h – file name completion:

Single Step – A user single steps through the history to the desired command.

Recall & Step – A user recalls how recent the command is and single steps to it.

History completion on command name – A user types a command name and presses the *history completion key*. The system searches the history for the most recent command matching the given name and presents it to the user. If the matching item is not the desired one, the user presses the *history completion key* again and the system finds the next most recent command matching the given name. This is repeated until the desired command is presented or no more commands match.

Constraints of the Study

Our primary research objective is to explore the issue of user effort in the specification of a recurrent command using cognitive modelling as the investigative methodology and not the development of the methodology. Thus, our study is subject to the limitations that exist with the cognitive modelling approach.

One consequence of these limitations is that we are unable to model the task of command generation or the task of deciding which method and design to use for command specification because these tasks are not modelled by the methodology. Despite this, there is value in studying the command specification task independent of the command-generation and decision-making concerns. In particular, we can explore the advantages and disadvantages of methods involving the use of history tools and typing.

A second consequence of these limitations is that we only model expert, error-free, user behaviour associated with command specification. However, there is value in such a study because we can examine the issue of user effort under optimal conditions.

A third consequence of these limitations is that parallel processes are modelled either as a series of processes or as a single process (i.e. the one requiring the longest time to completion); the serial approach is simpler to implement but leads to conservative predictions of task performance times while the one-parallel-process approach is difficult to implement but more accurate. The one-parallel-process approach is based on John (1988)'s proposal of using the critical path technique to model parallel processes. The technique involves specifying the component processes, their duration, and their inter-dependencies to form a network and then determining the path through this network that takes the longest time. We opted to use this critical path approach but we did not explicitly formulate the network in the course of generating our models; we simply selected the process from the group of parallel processes that was considered to be the critical one. Our informal use of the critical analysis approach (i.e., do not detail insignificant parallel processes) is acceptable (but perhaps not as thorough in the documentation of these parallel processes) as long as the predictions are

reasonably accurate and we adhere to two important criteria of cognitive modeling. These two criteria are: a) adequate coverage of the total task (i.e., take into account all first-order effects observed in the task domain) and b) approximation of the underlying processes (i.e., cognitive models, as an engineering prediction tool, need only include sufficient detail to do the design job) [Card, Moran, & Newell, 1983; John, 1988].

Aside from our study's constraints, there is also the concern regarding our claim that history-based, command-specification is a routine cognitive skill. Three arguments are made in support of this claim. First, a characteristic of skilled routine performance is the absence of learning. This occurs when task performance improves, as a result of learning, over many repeated encounters with a task until it reaches a plateau representing skilled behaviour. The highly repetitive nature of user interactions (recall the observed recurrence rate of 75%) suggests that there are plenty of opportunities for issuing recurrent commands and using history tools. Hence, users can become skilled with history-based command specification. Second, another characteristic of skilled routine performance is the routine nature of a task. That is, people generally become skilled in whatever becomes routine for them [Card, Moran, & Newell, 1983]. Since the subtasks (e.g., menu selection, search of a menu with a known organization, keystroke, recognition of a target item) associated with history-based command specification are prevalent in many of today's user interfaces, they are a routine part of user interactions and users can become skilled with such subtasks. Finally, the frequency with which user interactions are repeated suggests that history usage can also become routine. Therefore, in the context of our particular command specification task, history use can be considered a routine cognitive skill.

Estimates for Predictions

Our study uses a number of *MHP operators* and *application-specific parameters*. Each MHP operator is a specific elemental act of one of the three MHP processors (e.g., M_K is the keystroke operator associated with the motor processor M). Each application-specific parameter characterizes an aspect of the UNIX command specification task (e.g., \bar{n}_0 is the average number of characters in a UNIX command name). The following subsections enumerate these operators and parameters.

MHP Operators

Table 1 lists the durations of the various MHP operators used in our analysis. Three of the MHP operators were not available in Card, Moran, and Newell (1983)'s original list and had to be obtained elsewhere: menu search, associative memory retrieval, and menu selection with a mouse.

Menu Search. We estimated the menu search operator from an experiment (see Appendix D) because there were no prior estimates. The experiment examined the task of visually searching a menu ordered from most recent to least recent references of the menu's items. Our empirical data suggest that the time to search such a menu is approximated by a linear function of the position of the target item in the menu. We used the mean intercept and slope estimates obtained for the locality conditions (see Table 5 in Appendix D). The reason for focusing on

Parameters for Component Processes	Time (msec)	Source
Cognitive Operators		
C_E Execute a mental step	70	OO(240)
C_M Retrieve from memory	1200	John(44)
C_{CRT} Choice reaction time	340	CMN(74)
Perceptual Operators		
P_I Perceive an image	100	CMN(32)
P_E Eye movement	230	CMN(25-28)
P_{S_0} Menu Search – Intercept	690	Appendix D
P_{S_1} Menu Search – Slope	110	Appendix D
$P_S(l)$ Menu Search – Combined	$690 + 110 l$	Appendix D
Motor Operators		
M_B Press a button	70	CMN(69)
M_K Type a character		
Best typist (135 wpm)	80	CMN(264)
Good typist (90 wpm)	120	CMN(264)
Average skilled typist (55 wpm)	200	CMN(264)
Average non-secty typist (40 wpm)	280	CMN(264)
Worst typist	1200	CMN(264)
$M_{P_M}(l)$ Point to the l th item in menu	$-107+223\log_2(l+1)$	MSB(165)
$M_{D_M}(l)$ Drag to the l th item in menu	$135+249\log_2(l+1)$	MSB(165)
M_H Home to mouse or keyboard	360	CMN(237)

CMN(32)	[Card, Moran, & Newell, 1983] on page 32.
Appendix D	See Table 5, Appendix D.
John(44)	[John, 1988] on page 44.
MSB(165)	[MacKenzie, Sellen, & Buxton, 1991] on page 165
OO(240)	[Olson & Olson, 1990] on page 240.

Table 1: MHP Operators used in the user-effort analysis.

locality rather than non-locality behaviour is that our studies in Chapter 5 suggest that history tools for reuse are extremely useful during periods where clusters of user command line references recur (i.e., phases) as opposed to periods in which user command line references are shifting from one locality set to another (i.e., transitions).

Associative Memory Retrieval. A recent extension to the MHP/GOMS framework, prediction of user behaviour in command-abbreviation tasks, provided an estimate for the cost of retrieving a completely arbitrary association between a stimulus word and its required letter-combination [John, 1988]. Because several of our history-based designs involve an associative form of memory retrieval (e.g., actions for the history task, a number for a recurrent command, an argument from a recurrent command), we assume that this estimate is appropriate for our purposes. However, our memory retrievals are not necessarily arbitrary associations.

Thus, John (1988)'s estimate may be somewhat conservative for our study (i.e., it provides an upper bound). The appropriateness of using this estimate is examined later in an experiment.

Menu Selection with a Mouse. Fitts (1954) demonstrated that the time to move a distance A to acquire an object of size W varies with A and W :

$$MT = a + b \log_2(2A/W) \quad \text{Eqn. (1).}$$

Recently, MacKenzie (1989) noted that when A is small and W is large, the \log term in Eqn. (1) is negative and that this is theoretically unsound. Instead, MacKenzie (1989) proposed the following corrected formulation of Fitts' Law:

$$MT = a + b \log_2(A/W+1) \quad \text{Eqn. (2).}$$

Using this formulation, MacKenzie, Sellen, and Buxton (1991) empirically determined values for a and b for point and drag movement tasks involving a mouse. These results are used as estimates for our two kinds of mouse selection operators (see Table 1). The "point with mouse" operator involves moving a mouse to an object and selecting it (e.g., static menu selection) while the "drag with mouse" involves moving a mouse to an object with the mouse button held down and selecting it by releasing the button (e.g., popup menu selection).

We assume that the mouse is always located W units above the menu and that the height of each menu item measures W units. Therefore, to move to the l th item in the menu, the distance travelled is lW and the resulting log term is replaced by $\log_2(l+1)$.

Application-Specific Parameters

Table 2 lists the values for parameters characterizing UNIX command specification and history use. They include: probability of a command line having no argument, average number of characters in a command name and command argument, and average number of arguments. The values for all but two parameters are derived from an analysis of Greenberg (1988a)'s UNIX command traces. We elaborate on our choice of the values for the two exceptions.

The value for the probability of locating a command in each menu position (p_l) is chosen to simulate the locality behaviour and is identical to the one used in our menu search experiment. We did not use the same probability distribution observed in our studies of locality in Chapter 5. The reason is that the full range of locality set sizes (observed for real user sessions of 500 or more user commands) cannot be recreated in our two experiments (i.e., menu search and accuracy) in which user sessions were only 100 commands long. In addition to scaling down the length of a typical user session for the purposes of our experiment, we scaled down the locality set sizes that occur and we used a probability distribution that reflects the observed trend in the distribution of these 8 locality set sizes (i.e., a large number of locality sets of size 1 to 4). For the same reason as that presented for the menu search operator (see Appendix D), our study focuses on locality rather than non-locality behaviour.

The value for the average number of command lines with the same command name in a locality set (e) is derived as follows. Our analysis of Greenberg (1988a)'s UNIX user traces indicates that no more than three command lines beginning with

Parameters for Characteristics of UNIX Command Specification & History Usage	Values Mean \pm SD
Probability of Command Line p_a With no arguments	.37 \pm .19
Probability of search item at menu position $l=1$ to 8 p_l Small locality sets	.26,.24,.22,.20,.02,.02,.02,.02
Average Number of e Similar command lines examined a Command arguments	2 1.56 \pm .52
Average number of characters in \bar{n}_0 Command name \bar{n}_A Command arguments	3.28 \pm .57 5.31 \pm 1.37
Average Number of Menu Items Examined $\sum_{l=1}^8 lp_l$ Small locality set	2.72

Table 2: Table of estimates related to characteristics of UNIX command specification and UNIX history usage required for our current study.

the same command name occur within the same locality set. We did not examine the user traces to derive the exact probability distribution because it would involve a great deal of work. Instead, we assumed that all three cases occur with equal probability. Therefore, e is the expected value of observing 1, 2, and 3 occurrences of a command line with the same command name in a locality set (i.e., $e = E(s) = \sum_{s=1}^3 sp(s) = 2$).

Procedure

Using MHP as a characterization of human information processing, we generate a description of the MHP operations involved in the command-specification task for each proposed design (i.e., we generate a cognitive model). The resulting task description appears in Appendix E. The user effort associated with each proposed design is the sum of the operations in the associated cognitive model (see Appendix E). Table 3 contains the user effort equations obtained by substituting the values for the application-specific parameters.

Our analysis assumes that typing is the default course of action to take for command specification. Therefore, whenever a history tool is used, the user must switch from the default method (i.e., typing) to the history-based method. In all the history-based designs, this switching operation is manifested as an additional mental operation to retrieve the actions for the design from long-term memory. This memory retrieval requires a longer cognitive cycle C_M and represents the penalty for switching to a history-based method from the default typing method. Our assumption of a default method (i.e., typing) and thus a penalty for using the

Proposed Designs	Mental Effort	Physical Effort	System Effort
Typing	$17.5C_E$	$10.5M_K$	
Command Feature			
!!	$2C_M + 3C_E$	$3M_K$	
!name	$C_M + 7.3C_E$	$5.3M_K$	
!number	$2C_M + 5C_E$	$5M_K$	
!?argument	$2C_M + 9.3C_E$	$8.3M_K$	
History menu			
Popup	$C_M + 3C_E + \sum_{l=1}^8 p_l P_S(l)$	$2M_H + \sum_{l=1}^8 p_l M_{D_M}(l)$	S
Static	$C_M + 3C_E + \sum_{l=1}^8 p_l P_S(l) + P_E$	$2M_H + \sum_{l=1}^8 p_l M_{P_M}(l)$	
Step Buffer			
Single step	$C_M + 4.7C_E + P_{S_0} + 2.7P_{S_1} + 2.7P_I$	$3.7M_K$	$2.7S$
Recall & step	$2C_M + 5.7C_E + P_{S_0} + P_{S_1} + P_I$	$3.7M_K$	S
Command name with history completion	$C_M + 9.3C_E + 2P_{S_1} + 2P_I$	$6.3M_K$	$2S$

Table 3: The user effort associated with each command-specification design after substituting for the values for the application-specific parameters. The user effort is partitioned into mental, physical, and system effort components. Note, in order to simplify the notation, we use S to represent system response time. However, the value for S may be different for different designs because they may take different amounts of system time to perform.

history-based method is reasonable because, instead of providing the command directly, users are performing an additional task (i.e., using an abstraction vehicle like history tools to produce the command).

The primary focus of our analysis is to examine the mental and physical effort involved in each design proposal. To facilitate this analysis, we partitioned the user effort associated with each design into the mental and physical components as well as a system effort component (see Table 3). The mental effort component contains all cognitive and perceptual operators for a design. The physical effort component contains all simple motor, menu selection, and keystroke operators for a design. The system effort component contains all user pauses, in a design, for system response S . Note, in order to simplify the notation, we use S to represent system response time in all the designs. However, the value for S may be different for different designs because they may take different amounts of system time to perform. In our analysis, we make sparing reference to the system effort component and in most cases, we ignore this component.

Cognitive models are primarily tools for predicting user behaviour but they are also useful as “tools for thought” [Newell & Card, 1985]. That is, MHP/GOMS embodies a theory of human information processing and in the course of doing the cognitive modelling, user interface designers are made aware of the human information processing requirements of a particular design in the form of the number and variety of MHP operators involved. However, cognitive models as tools for thought have been underrated especially in light of the fact that current cognitive models and their predictions have many limitations and are subject to further improvements. Because current cognitive models are still in need of further development, it is premature to place most of the attention on the predictions derived from cognitive models (e.g., Gray, John, Stuart, Lawrence, and Atwood (1990) and MacLeod and Tillson (1990)) and to place little attention on the insights derived from cognitive modelling. Thus, as part of our findings, we also present the insights derived from our analysis. Furthermore, we use the task performance times and MHP estimates to aid in the comparison of designs. In particular, numerical estimates (see Table 4) are used to quantify the user effort required in each design.

Proposed Designs	Predicted Times (msecs.)		
	Mental Effort	Physical Effort	System Effort
Typing	1220	$10.5M_K$	
Command Feature			
!name	1710	$5.3M_K$	
!number	2750	$5M_K^1$	
!?argument	3050	$8.3M_K$	
!!	2610	$3M_K$	
History Menu			
Popup	2399	$720 + 577$	S
Static	2629	$720 + 289$	
Step Buffer			
Single step	2790	$3.7M_K$	$2.7S$
Recall & step	3700	$3.7M_K$	S
Command name with history completion	2270	$6.3M_K$	$2S$

Table 4: The user effort equations in Table 3 after substituting for the all MHP operators except M_K . Note, unlike the other designs in the command feature category, the !name design does not require any mental retrievals including the mental retrieval of the command name because the command name is already in working memory. Note also, in order to simplify the notation, we use S to represent system response time in all the designs. However, the value for S may be different for different designs because they may take different amounts of system time to perform.

Results and Discussion

Our comparative analysis of command specification designs is divided into three parts. The first part compares the user effort associated with each history-based design class: *command feature*, *step buffer*, and *history menu*. The second part compares user effort across the three history-based design classes by examining the advantages and disadvantages of using recognition versus recall versus a combination of recognition and recall. The third part compares the use of *typing* against the use of history tools for specifying a recurrent command.

Human Performance within a Class of History Designs

Command Feature Designs. While design *!!* is less powerful than design *!number*, they are comparable in terms of the mental and physical operations that are involved (see Table 3). They both require two memory retrievals: one for the method and the other for the command number or an association between the generated command and the previous command. However, design *!number*¹ takes more time with increasing number of digits in a command number because of additional cognitive cycles C_E and keystroke operators M_K . Because the predicted times for the two designs are comparable and design *!!* takes slightly less time, the balance of the analysis uses design *!!* to represent both designs.

Designs *!name* and *!?argument* represent two different ways of recalling a recurrent command: by name or by argument. There are two reasons why *!?argument* takes more time than design *!name*. First, UNIX command arguments are longer than command names (compare \bar{n}_A and \bar{n}_0 in Table 2) and this results in additional cognitive cycles C_E and keystroke operators M_K . Second, an extra memory retrieval C_M is required to choose a command argument which uniquely retrieves the command. Such a retrieval is not required in design *!name* as the name is available in working memory. The additional cognitive complexity in design *!?argument* may explain why only 2 of the 5 subjects in our exploratory study made use of the *!?argument* feature while all 5 subjects used the *!name* feature (see Table 2a in Chapter 4).

History Menu Designs. The *popup menu* design requires system response and drag-with-mouse operators while the *static menu* design requires eye-movement and point-with-mouse operators (see Table 3). The *static menu* design requires slightly more mental effort because of eye movement (see Table 4).

While both our mouse selection tasks involve moving a mouse to acquire an object, the tasks have different task difficulty levels². MacKenzie, Sellen, and Buxton (1991) report an index of difficulty of .25 secs/bit. for the drag-with-mouse operation as opposed to .22 secs/bit for the point-with-mouse operation. Also, for an 8-item menu, a drag-with-mouse operator takes, on average, twice as long as a

¹ In any intensive user session (i.e., more than 99 commands), a command number should, on average, contain 3 digits. When a command number has only one digit, the two designs are identical.

² The *log* term in Eqn (2) is known as the index of difficulty and carries the units “bits” (because the base is “2”). The reciprocal of the coefficient b is the index of performance which is the human rate of information processing for the movement task.

point-with-mouse operator (see the second value in physical effort in Table 4). We conjecture that the drag-with-mouse operation is more difficult compared to the point-with-mouse operation because of the tension involved in holding down the mouse button as the mouse is moved. Without even considering the *S* operator in the *popup menu* design, the time to perform the *static menu* design is slightly faster than the *popup menu* design (see Table 4).

Other design considerations point favourably to the *static menu* design. In particular, it provides continuous user feedback of a history's contents because it remains on a user's screen while *popup menu* users must bring up a menu to glance at its contents. Furthermore, a drag-with-mouse operation in the *popup menu* design has more potential for user error due to failure to hold the mouse button down while making a selection. On the other hand, a *static menu* does incur screen real estate costs but such costs can be minimal because a typical locality-set-based history menu contains one to five items (see Table 1 in Chapter 5). However, the menu may be obscured by other overlapping windows and users need to perform extra operations to unveil the menu. Nevertheless, the advantages of feedback, less errors, and less effort outweigh these disadvantages. Therefore, the balance of our analysis will use the *static menu* as the prototypical *history menu* design.

Step Buffer Designs. In all three *step buffer* designs, a user must pause after each stepping operation to wait for the system to respond (*S*) with the retrieved command (see Table 4); the number of pauses is proportional to the average number of items examined prior to a selection. There are more pauses in the *single step* and *command name with history completion* designs than in the *recall & step* design.

On the other hand, in the *recall & step* design, a user must accurately recall the position of a desired command and single step directly to it. The extra mental retrieval operation in this design results in more mental effort (3.7 secs.) compared to the other two step buffer designs (2.3 secs. and 2.8 secs.).

The *command name with history completion* design requires more physical effort than the other two designs because of the additional keystrokes required to type a command name. However, it requires less mental effort compared to the other two designs. In general, minimizing the use of limited cognitive resources and thus mental effort is important as it ensures that these resources are available for other tasks. Assuming average or better typing skills ($M_K \leq 200$ msecs.), the *command name with history completion* design is a faster design.

Human Performance between Classes of History Designs

Pure Recall Designs. All *command feature* designs and the *recall & step* design are pure recall designs; they exploit a user's ability to recall characteristics of recurrent commands. Design *recall & step* requires more mental effort (see Tables 3 & 4) than all three *command feature* designs because of the perceptual elements of the design (e.g., confirmation of a system retrieval prior to selection). However, design *!argument* requires more physical effort than any of the other pure recall designs because it requires more keystroke operators (8.3 see Table 4). Note, an alternative to the *!argument* design would be to choose a substring to identify the *argument* which would require fewer keystrokes but it would be at the

expense of more mental effort associated with determining the appropriate substring (i.e., an extra C_M operator); the additional mental effort with the *!substring_of_argument* design would be more than the savings in physical effort. Because of the considerable physical and mental effort required in the *!argument* and *recall & step* designs, we ignore them in the subsequent analyses.

Pure Recognition Designs. All *history menu* designs and the *single step* design are pure recognition designs; they exploit a user's perceptual/recognition abilities by providing a visual interface to a user history. The *static menu* design requires less mental effort than the *single step* design (2.6 secs compared to 2.8 secs.). If users are average or better typists ($M_K \leq 200$ msec.), the *single step* design requires less physical effort compared to the *static menu* design (1.0 secs. compared to .7 secs) because single-step cursor operations are faster. However, users with poor typing skills ($M_K \geq 280$ msec.) will be faster with the *static menu* design rather than the *single step* design because the homing-to-device and menu selection operations are faster. Also the *single step* design requires system response time, once for each item examined, while none are required for the *static menu* design. This is not an issue on fast systems.

Pure Recall versus Pure Recognition Designs. The *!name* recall design requires less mental effort than either of the two recognition designs because its associated retrieval operation involves a working-memory retrieval C_E which is simpler than the visual search operations $P_S(I)$ in the two recognition designs (see Tables 1 & 3). However, the three recall designs – *!*, *!argument* and *recall & step* – require more mental effort than the two recognition designs because they involve long-term memory retrieval C_M (see Tables 1 & 3).

All pure recall designs and the *single step* recognition design require typing skills (see Table 4) while the history menu designs require skill with a mouse. The physical effort in the *static menu* design (1 sec.) is more than the physical effort in the recall designs when users are better than average typists ($M_K < 200$ msec.). However, if typing skills are poor ($M_K \geq 350$ msec), the *static menu* design requires less physical effort than all the recall designs.

Assuming that system response time is negligible, $S = 0$, Figures 1 and 2 plot the time to perform each proposed design as a function of keystroke operator times M_K ; Figure 2 is Figure 1 extended to include a worst case typist ($M_K = 1200$ msec). From Figure 2, we observe that the *static menu* design is superior to all recall designs when a user's typing speed is poor ($M_K \geq 400$ msec).

A Combined Recall and Recognition Design. The *command name with history completion* design is a combined recall and recognition design; it exploits recall as well as perceptual and recognition abilities. Users must provide the command name as a selection pattern and must inspect the retrieved command to accept or reject it. While the memory retrieval time is small, because the command name is available in working memory, this design requires perceptual, cognitive, keystroke and system response operators. In general, this combined recall and recognition design provides no substantial benefits compared to the pure recall or pure recognition designs.

Compared to the pure recall designs, this design is neither superior in cognitive (2.3 secs.) nor physical effort (6.3 M_K). In fact, the *!name* design and this design are similar with respect to retrieval of a command name but the *!name* design does not require the extra perceptual, cognitive, motor, and system response

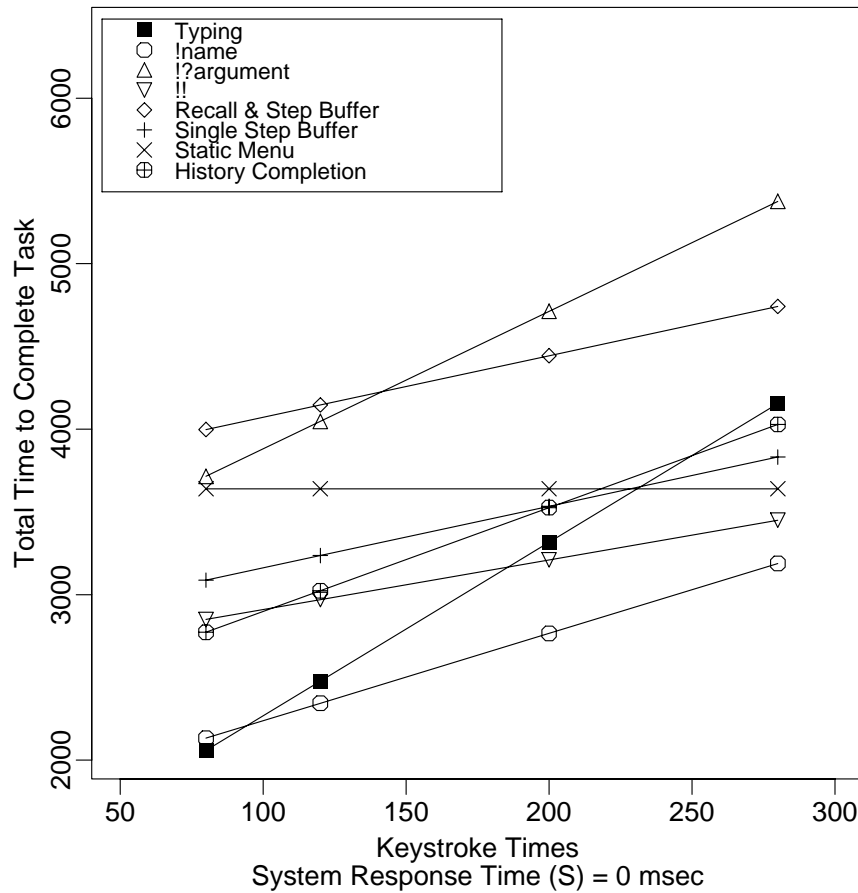


Figure 1: The time to perform each design is plotted as a function of keystroke times ($M_K = 80, 120, 200,$ and 280 msec.) The pure recall and pure recognition history designs are denoted by unfilled graphical markers and crosses, respectively. The combined recognition and recall history design is denoted by an unfilled graphical marker with a cross in the center. The *typing* design is denoted by the black box.

operations (see Table 3). When a user's typing skill is poor, all pure recall designs, except for *!?argument*, require less time than this design.

Compared to the two pure recognition designs, this design requires less mental effort. However, users with average to worse typing skills ($M_K \geq 200$ msec.) require more physical effort to use this design (see Figure 2). If system response time ($S > 0$) is considered, the performance time for this design will be higher.

Typing versus Using a History Tool

Typing requires the least mental effort compared to any history method (see Table 4) because we assume that the command is already in working memory and

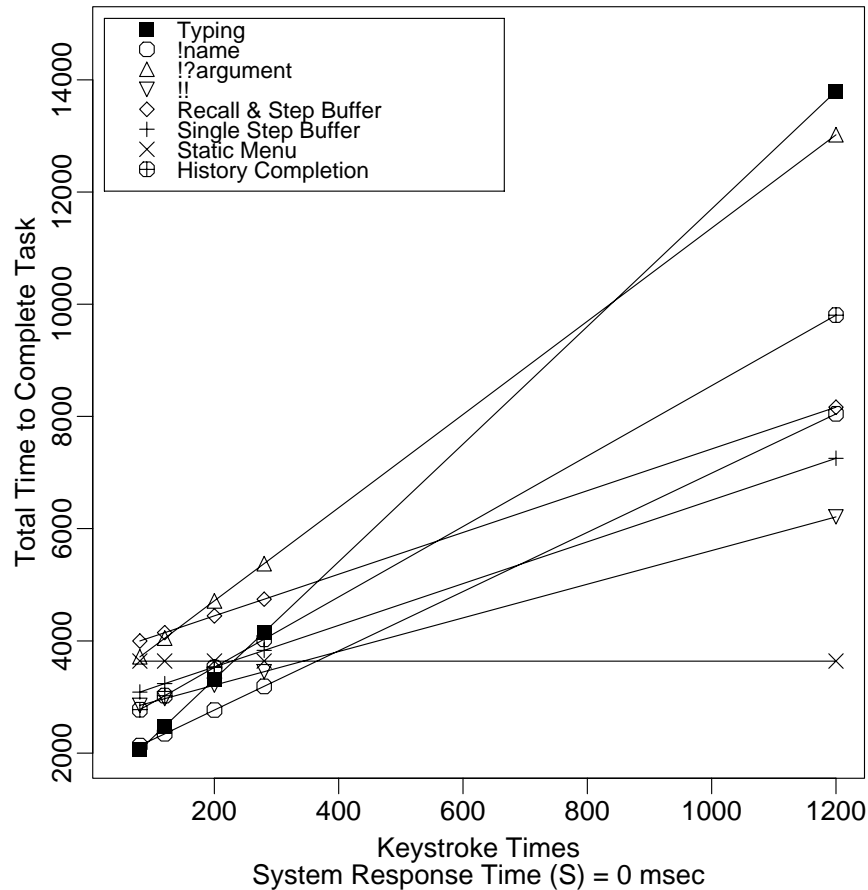


Figure 2: Figure 1 extended to include a non-typist keystroke time (i.e., $M_K = 1200$ msec.). The time to perform each design is plotted as a function of keystroke times ($M_K = 80, 120, 200, 280,$ and 1200 msec.). The pure recall and pure recognition history designs are denoted by unfilled graphical markers and crosses, respectively. The combined recognition and recall history design is denoted by an unfilled graphical marker with a cross in the center. The *typing* design is denoted by the black box.

neither long-term memory retrieval nor visual search are involved. There are only cognitive operations associated with the selection of command line words from working memory and the initiation of motor operations for keystrokes M_K . As well, *typing* requires no point-and-select and no system response times. However, it requires the largest number of keystroke operators (10.5 – see Table 4). As a result, this method is the worst for issuing a recurrent command when non-typists ($M_K = 1200$ msec.) are involved (see Figure 2). If users are expert typists ($M_K = 80$ msec.), this method is the fastest. However, in the case of non-expert typists, there is always at least one history-based design faster than *typing* despite the fact that all history-based designs require a memory retrieval for the history method ($C_M = 1200$ msec.).

The *lname* design is faster than *typing* and all other history designs when users have good to average non-secretarial typing skills (120 msec. (i.e., 90 wpm) $\leq M_K \leq 280$ msec. (i.e., 40 wpm) – see Figure 2). This occurs because it requires the least mental effort (1.7 secs.) of all history-based designs and requires half the physical effort of *typing* ($5.3 M_K$ compared to $10.5 M_K$).

The fastest recognition design, *static menu*, is the third fastest history design for users with average non-secretarial typing skills ($M_K = 280$ msec. (i.e., 40 wpm)) and is the fastest history design for users with worse-than-average, non-secretarial, typing skills ($M_K > 280$ msec. (i.e., 40 wpm) – see Figure 2). Furthermore, the combined recognition and recall design, *command name with history completion*, is faster than *typing* for users with worse-than-average, non-secretarial, typing skills ($M_K > 280$ msec. (i.e., 40 wpm) – see Figure 2).

Accuracy of the Cognitive Models' Predictions

An important concern with modelling user behaviour (i.e., typing commands and using history tools) in a different task domain is whether the two GOMS models support the underlying activities and whether the cognitive model predictions are reasonably accurate. While some of the activities considered in our study are those used in the development of the two GOMS models (Keystroke Level Model and GOMS Model of Immediate Behaviour), there is a new activity (i.e., menu search) whose estimate had to be obtained.

Also, in the interest of keeping both GOMS models simple, the details of mental operations like long-term memory retrieval [John, 1988] and “mentally preparing” to execute physical operators [Card, Moran, & Newell, 1983] were not elaborated. A mental operator is used to represent such operations. As described earlier, we assume that the time parameter of such operations are estimated by the mental retrieval operator that John (1988) determined. However, it is unclear whether this is a reasonably accurate estimate of the time to perform the retrieval activities that are part of the history-tool method.

In order to resolve the two accuracy concerns (i.e., being able to model and predict history-tool activities and to use the current estimate of mental retrieval time for our study), we selected a subset of the designs examined in our study and ran an experiment to collect actual task performance times. Then we compared the cognitive models' predictions against that of the observed performance times to assess the predictive accuracy of the models.

Method

The four selected designs are the *typing* method and one design from each of the three history-tool classes: *recall by argument*, *popup menu*, and *single step buffer*. All subjects performed all four tasks a number of times with the tasks being administered in a random order. The reason for combining the trials of all four tasks instead of blocking the trials by task is to mimic a typical user session in which the particular task used to issue a recurrent command is not known *a priori*.

Subjects

18 paid students and staff participated in the experiment. All subjects had knowledge and experience with UNIX, had previous experience using a mouse, and had varying typing skills ranging from touch typists to two-finger typists.

Materials and Apparatus

A nine-item history list was used. Items were organized from the most recent to the least recent. In the popup-menu task, the eight most recently referenced items were visible in the menu. The most recently referenced item appeared at the top of the menu and the 8th most recently referenced item appeared at the bottom of the menu. The stimuli were UNIX command lines and are listed in Table 5. For reasons similar to those in the menu search experiment, menu search features like item length and distinguishing characters (e.g., #) were either neutralized or absent.

The experiment made use of a SUN 3/50 workstation consisting of a high-resolution graphic display with a mouse and keyboard. A simple window manager – MGR – from Bellcore was used to program the experiment. The SUN display was divided into three windows. The top window contained instructions to the subjects. The bottom window displayed an error message whenever a subject made a mistake (e.g., selecting an inappropriate key on the keyboard). The middle window was the primary interaction window in the experiment. A stimulus was presented to a subject in this window and a subject's response was cued and initiated through this window.

Tasks

The experimental task is a *test* task consisting of two subtasks. Subjects were presented with a UNIX command line whose words were jumbled and they

Practice 0	Practice 1	Real Test
tail -5 other	rm -i bkup/*	lpr -Plw *.c &
head -5 Msgs	cp bkup/* ../	dbx main core
cat m* > Msgs	mv mbox bkup	mv a.out main
cat msg1 msg2	ls -alt mbox	cc -c main.c &
tail -f save	pwd; du -s .	co -r4 main.c
ls -l > save	rm -rf srmdir	lint -h test.c
mv msg4 save	ls -lt srmdir	co -r2 test.c
mv msg0* dir	du srmdir p	rlog -l test.c
ls -a dir p	alias p more	make -f clean

Table 5: The menu items used in the validation experiment.

had to unjumble it to form a legal UNIX command line³. This *unjumbling* task simulates command generation with the result that the unjumbled command is cached in a subject's working memory. After unjumbling the command, subjects pressed the "carriage return" key. Then, subjects were cued to perform one of the four command specification tasks: TYPE, PATTERN, STEP, or MENU <please use the mouse>. The TYPE cue refers to the *typing task* which involved typing out the whole generated command line followed by a "carriage return". In the other three cued tasks, subjects retrieved the generated command from a history using the cued history task.

The *recall-by-argument* task involved mentally selecting any word other than the first word of the generated command line and using it as a retrieval pattern. The retrieval is specified by typing the characters '!', followed by the argument and terminated with a 'carriage return'.

The *single-step-buffer* task involved stepping through the history list an item at a time. To step to the next most recent history item, the 'j' key is pressed. To step to the previous most recent history item, the 'k' key is pressed. These keys are typically alternative forward/backward cursor keys in most UNIX applications. When the desired command appears, selection is made with a 'carriage return'.

The *popup-menu* task involved selecting the generated command from an eight item menu. The menu pops up when the middle mouse button is held down. A menu selection is made by dragging the mouse cursor down to the item, while holding the middle button down, and then releasing the middle button. If the generated command does not appear in the menu, a null selection is made by releasing the mouse button while the cursor is positioned either above or below the menu area.

Procedure

Prior to the start of an experimental session, subjects performed a typing speed test. Each *session* consists of two *practice blocks of test trials* and a *real block of test trials*. The first practice block consists of 4 sets of 9 trials each. In the first set, subjects are presented with each stimulus in the practice block to unjumble and type-in – one at a time from the least recent to the most recent. If an error is made, they repeat this unjumbling and typing task. In the other three sets, subjects practice each of the three history tasks independently. Note, each subject was randomly assigned to a practice order with an equal number of subjects for each practice order.

The second block of trials gives a practice run of the real thing. Subjects are shown each item in the stimulus set, from the least recent to the most recent, in a combined unjumbling and typing task. Then, subjects work through 48 test tasks; 21 of which are typing and the remaining 27 are equally divided amongst the three history-based command specification tasks. The particular cued task is randomly assigned.

³ Ambiguities relating to this unjumbling step are minimized but some UNIX commands can be ambiguous in the absence of a real task context. Therefore, we tried to choose arguments that would suggest what the appropriate unjumbled command should be (e.g., cat file1 file2 rather than cat file2 file1).

The real block of test trials follows the same procedure as the second practice block but there are 87 test tasks; 39 of which are typing and the remaining 48 are equally divided amongst the three history-based tasks.

Results and Discussion

For each of the four tasks, we generated a task description (see Appendix F). Table 6 lists the predicted performance time associated with each task. Except for S and M_K (determined empirically for each subject), all MHP operators were obtained from the literature (see Table 1). Each subject's keystroke time, M_K , was obtained from a timed typing task (i.e., typing time is divided by the number of characters they typed from a 471 character piece of text). The mean keystroke time was 270 ± 20 (18)⁴ msec. The system response time, S , was measured in the trials for the pop-up menu task. The mean system response time was 220 ± 4 (278) msec.

In each trial, one of the four tasks is randomly assigned to the subjects. As a result, part of the performance time includes a *choice reaction time*. This is the time required to react to the requested task by initiating the correct task response sequence. Since there are four possible tasks from which the correct response is chosen, reaction time is a function of the time to make one of the four possible choices. According to Card, Moran, and Newell (1983), this choice reaction time is:

$$C_{CRT} = 150 \sum_{t=1}^4 p_t \log_2(1 + 1/p_t)$$

where p_t is the probability of an occurrence of task t or equivalently, the fraction of the trials containing task t . Each of the three history-based tasks appears in 16 trials while the typing task appears in 39 trials for a total of 87 trials.

Tasks	Predicted Times
Typing	$P_I + C_{CRT} + (4 + 3a + \sum_{A=0}^a n_A)C_E + (1 + a + \sum_{A=0}^a n_A)M_K$
Recall by argument	$P_I + C_{CRT} + 2C_M + (n_A + 4)C_E + (n_A + 3)M_K$
Popup menu	$P_I + C_{CRT} + C_M + 2C_E + M_H + M_{D_M}(I) + S + P_S(I)$
Single step buffer	$P_I + C_{CRT} + C_M + P_S(I) + IP_I + (I + 2)C_E + IM_B + M_K$

Table 6: The predicted task performance time as a function of the relevant MHP operators. a is the number of arguments in a UNIX command. n_A is the number of characters in word A of the UNIX command. I is the serial position of the target item. S is the system response time.

⁴ The notation 270 ± 20 (18) represents a mean value of 270, a standard error of 20, and a sample size of 18.

A subject's real test trial was discarded if an error was committed in the course of the trial (e.g., incorrect selection, mistyped argument or command name). Each experimental task has a number of different instances related to the parameters of the task. These parameters are obtained from the equation for the predicted times to perform each task (see Table 6).

Typing task: a is the number of arguments in the command line and $\sum_{A=0}^a n_A$ is the number of characters in the command line (excluding spaces).

Recall by argument: n_A is the number of characters in the argument.

Popup menu: l is the menu position of the desired command line (note: $l = 9$ represents an item not on the menu).

Single step buffer: l is the number of steps to the desired command line.

Table 7 lists the various instances of each experimental task in terms of the task parameter values. All subjects' trials were separated into the appropriate task instances. Table 7 lists the total number of trials and error-free trials associated with each task instance. Figure 3 plots the mean observed times for the 27 instances of the 4 tasks as a function of the predicted times. The $y = x$ line provides a reference for comparing the mean observed time to the predicted time.

The measure used to describe the error in the prediction is $e_o(i)$:

$$e_o(i) = \frac{\bar{O}_i - P_i}{\bar{O}_i} \times 100\%$$

where P_i is the predicted time for the i th task and \bar{O}_i is the mean observed time for the i th task. The magnitude of the difference in observed and predicted times is expressed as a proportion of task performance time (i.e., observed time). This is a more meaningful measure than the absolute difference since different tasks have different task complexity and hence different task performance times.

Our reason for expressing the error in terms of the percentage deviation from the observed time rather than the percentage deviation from the predicted times – Card, Moran, and Newell (1980)'s approach – is that the error measure provides a more accurate assessment of the deviation from the value of interest (i.e., observed time) and hence, the accuracy of the predictions made by a model. John (1988) used the $e_o(i)$ metric in her analysis of the accuracy of the cognitive models generated using the GOMS Model of Immediate Behaviour.

A goal of engineering models is to make predictions averaging within 80% of actual performance. As the data in Table 8 indicate, the prediction errors in 21 instances of the 4 tasks are well within the 20% error criteria. Of the 6 task instances having greater than 20% prediction error, 1 of them is associated with the single-step-buffer task (i.e., $l = 1$). We conjecture that because of an experimental bias for items at the top of the history list, users have more familiarity with recent command lines and are more adept at handling a history retrieval involving the most recent command line. This skill coupled with a simple response scheme (single stepping using cursor keys) results in a faster task performance time compared to the task performance times of items further down in the history list. This conjecture is supported by a similar result in our menu search

Tasks & Parameters	Values of Parameters for Task Instances	Number of Trials	Number of Error-free Trials
Typing, ($a, \sum_{A=0}^a n_A$)	(2, 11), (2, 12), (3, 11)	702	673
Recall by argument, n_A	1, 2, 3, 4, 5, 6	288	261
Popup menu, l	1, 2, 3, 4, 5, 6, 7, 8, 9	288	278
Single step buffer, l	1, 2, 3, 4, 5, 6, 7, 8, 9	288	269

Table 7: For each experimental task, the task instances are enumerated in terms of the task parameter values. Also, the total number of trials and the number of error-free trials for each experimental task are presented. Note: In the menu task, an l value of 9 represents an item not appearing on the menu.

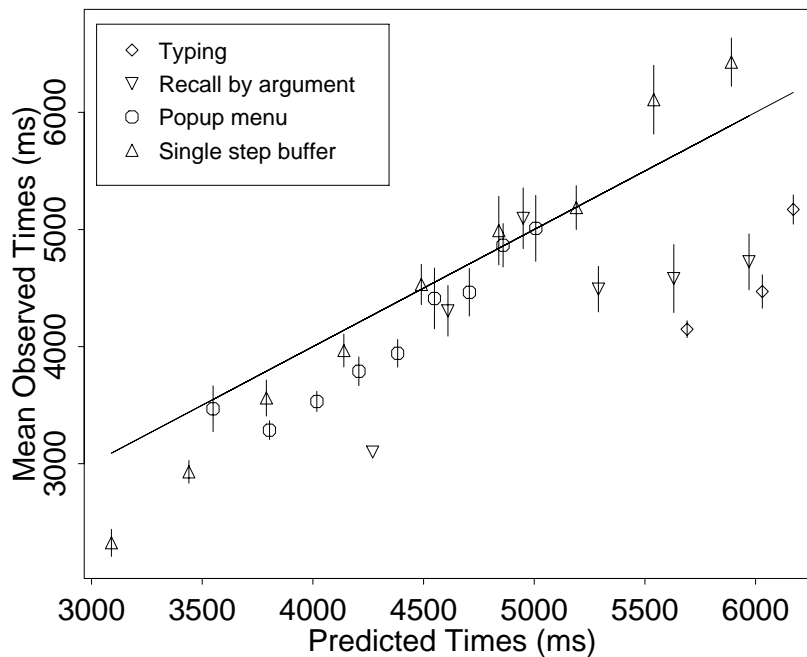


Figure 3: Plot of the mean observed times, along with error bars, as a function of the predicted times. The line is a $y = x$ line.

experiment (see the $l = 1$ case in Figure 4 of Appendix D). This anomalous behaviour is not evident in the popup-menu task because any user skill with recognizing the most recent item is offset by the less than optimal menu selection behaviour associated with a more complex response scheme. Note, we observed that subjects occasionally made adjustments in the drag operation for over/under-shooting the target. Also, the menu search experiment and the single-step-buffer design use a simple push-button response scheme.

Task	Predicted Times (msecs.)	Mean Observed Times (msecs.) Mean \pm SE (N)	% Error $e_o(i)$	>20% Error
Typing				
2 args, 11 chars	5690	4150 \pm 70 (361)	-37%	†
2 args, 12 chars	6030	4470 \pm 140 (144)	-35%	†
3 args, 11 chars	6170	5170 \pm 130 (168)	-19%	
Recall by argument				
1 character	4270	3100 (1)	-38%	†
2 characters	4610	4310 \pm 220 (65)	-7%	
3 characters	4950	5100 \pm 260 (51)	3%	
4 characters	5290	4490 \pm 190 (62)	-18%	
5 characters	5630	4580 \pm 290 (41)	-23%	†
6 characters	5970	4720 \pm 240 (41)	-26%	†
Popup menu				
position 1	3549	3470 \pm 200 (35)	-2%	
position 2	3804	3290 \pm 80 (54)	-16%	
position 3	4018	3530 \pm 90 (35)	-14%	
position 4	4208	3790 \pm 120 (34)	-11%	
position 5	4383	3940 \pm 120 (36)	-11%	
position 6	4549	4410 \pm 260 (18)	-3%	
position 7	4707	4460 \pm 200 (17)	-5%	
position 8	4859	4870 \pm 190 (18)	0%	
position 9	5007	5010 \pm 280 (31)	0%	
Single step buffer				
position 1	3090	2330 \pm 120 (35)	-33%	†
position 2	3440	2930 \pm 100 (36)	-17%	
position 3	3790	3560 \pm 150 (51)	-6%	
position 4	4140	3970 \pm 140 (36)	-4%	
position 5	4490	4530 \pm 170 (31)	1%	
position 6	4840	4990 \pm 290 (30)	3%	
position 7	5190	5190 \pm 190 (15)	0%	
position 8	5540	6110 \pm 290 (18)	9%	
position 9	5890	6430 \pm 210 (17)	8%	

Table 8: For each of the 27 instances of the four experimental tasks, the predicted times, the mean observed times (along with standard error and number of trials), the error as a percentage of mean observed times and predicted times, and the cases where the error is greater than 20% are presented. M_K is the mean keystroke rate 270 ± 20 (18) msecs.

The other 5 of the 6 task instances having greater than 20% prediction error are associated with the two tasks involving typing (i.e., typing and recall-by-argument). In all cases, the predicted times are longer than the observed times (see Table 8). We conjecture that the higher prediction error is attributed to an

inaccurate estimate of the keystroke time M_K . Note, in the interest of simplifying the processing of the subject's data, we decided not to capture the raw keystrokes used in the typing and recall-by-argument tasks. However, the MHP/GOMS framework only deals with error-free behaviour and we need to take into account the possibility that subjects might make typing mistakes and take time to correct them. We accounted for this time by measuring a keystroke time M_K (i.e., time per keystroke) that included a fraction of the time to detect and correct typing mistakes (i.e., we used the mean typing time – the time to type a 471 character piece of text divided by the number of characters in the subject's typed document). In hindsight, this typing task has a number of problems. First, typing a 471 character piece of text is somewhat different from typing a UNIX command line (11 or 12 characters long) or an argument (1 to 6 characters long). Second, there are more instances of typing mistakes, and hence more time spent recovering from typing mistakes, in a typing task involving a 471 character piece of text than in a 11 or 12 character command line. Thus, our estimate of M_K is inflated. It may have been more appropriate to have subjects perform a series of timed typing tasks involving UNIX command lines.

Table 9 lists the mean and range of the absolute prediction error for the 4 tasks by combining all instances of the same task. Both the popup-menu and

Tasks	Number of Tasks	Absolute % Error $e_o(i)$ Mean (Range)
Editing - CMN (404)	12	10% (0% – 37%)
Drawing - CMN (404)	15	19% (1% – 51%)
Computing - CMN (404)	5	24% (9% – 39%)
<i>Combined</i>	32	16% (0% – 51%)
Command abbreviation - John (36)	3	26% (0% – 71%)
Fitts & Seeger Experiment - John (39)	4	16% (2% – 33%)
Duncan Experiment - John (40)	4	8% (7% – 10%)
Morin & Forrin Experiment - John (42)	6	23% (13% – 38%)
<i>Combined</i>	17	19% (0% – 71%)
Typing commands	3	30% (19% – 37%)
Recall by argument	6	19% (3% – 38%)
Popup menu	9	7% (0% – 16%)
Single step buffer	9	9% (0% – 33%)
<i>Combined history designs</i>	24	11% (0% - 38%)
<i>Combined</i>	27	13% (0% - 38%)

Table 9: The absolute prediction error in 3 different types of tasks (editing, drawing, and computing tasks) examined by Card, Moran, and Newell (1980) (abbreviated as CMN with page numbers in brackets); in 4 different stimulus-response compatibility (SRC for short) tasks examined by John (1988); and 1 command typing and 3 different history tasks that we examined. In the case of Card, Moran, and Newell (1980)'s results, we recomputed the prediction error using $e_o(i)$.

single-step-buffer tasks have a mean absolute prediction error of less than 10% (i.e., 7% and 9%) while the recall-by-argument task has a mean absolute prediction error of 19%. The typing task is the only task that does not meet the 20% error goal.

All three history tasks involve at least one retrieval operation (i.e., retrieval of the task method – see Table 6). A comparison of the absolute prediction errors of the instances of these 3 tasks reveals a range of 0% to 38% and a mean absolute error of 11% (see Table 9). This result indicates that the estimate we used for C_M is an appropriate estimate for the mental retrieval of a task method. Only the recall-by-argument task has an additional mental retrieval operation for selecting the appropriate command argument for the retrieval pattern (see Table 6). Table 8 reveals that the predicted time in all instances of this task is higher than the mean observed time and the range of the absolute prediction error is 3% to 38%. However, because there is a possibility that an inflated estimate of keystroke time M_K may lead us to predict a higher task performance time, we are unable to assess the appropriateness of using the same mental retrieval operator for the second retrieval operation.

An important question is how good the predictions for our command specification task are compared to GOMS predictions for other tasks. We are aware of only two studies which examine the accuracy of the predictions made by the cognitive models generated using the MHP/GOMS framework.

The first study is the experiment to determine how well the Keystroke Level GOMS Model predicts performance times [Card, Moran, & Newell, 1980]. 32 different task-system combinations were examined in the experiment. 14 different tasks (4 editing, 5 drawing, and 5 computing) and 11 different systems (3 editors, 3 drawing packages, and 5 command executives) were considered in the experiment. The mean absolute prediction error for the editing, drawing, and computing tasks are 10%, 19%, and 24%, respectively (see Table 9 for the range of absolute prediction errors). The mean absolute prediction error for the combined group of tasks is 16%.

The second study is a series of experiments to determine how well the GOMS Model of Immediate Behaviour predicts performance times for a number of immediate behaviour tasks [John, 1988]. The group of tasks included a set of 3 command abbreviation recall tasks and 2 different sets of spatial SRC tasks (Fitts & Seeger experiment and Duncan Experiment) and a set of symbolic SRC tasks (Morin & Forrin Experiment) drawn from the literature. John (1988) generated predictions for response time performance for the 4 sets of immediate behaviour tasks and compared them to the observed response time (see Table 9). The mean absolute prediction errors for the 4 sets of tasks are 26%, 16%, 8% and 23% and the mean absolute prediction error for the combined set of tasks is 19%.

Our own experiment examined the accuracy of using both GOMS models (i.e., Keystroke Level Model and the GOMS Model of Immediate Behaviour enhanced to include an estimate for the menu search operator). The mean absolute prediction errors for the 4 tasks are as good if not better than the mean absolute prediction errors of the tasks examined by Card, Moran, and Newell (1980) and John (1988). Compared to the mean absolute prediction errors for the combined group of tasks examined in each of the other two studies (i.e., 16% and 19%), our combined group of tasks has a mean absolute prediction error of 13%.

Remarks about Our Studies

In closing, we make two sets of general remarks concerning the problems of cognitive modelling and the possibilities for future work.

Problems with Cognitive Modelling

The MHP/GOMS framework assumes expert, error-free, routine cognitive-skilled user behaviour and stipulates that predictions are *approximate*. Typically, users are rarely true experts and their behaviours are rarely error-free. This was especially evident in the typing task in our experiment where only one subject was a fast and accurate touch typist. In many instances of the test trials, subjects were observed to be correcting their typing mistakes. Typing errors may be, in part, caused by the fact that subjects were unfamiliar with the keyboard and its layout. Thus, the predicted times represent a lower bound because the observed performance times will typically include non-error-free or suboptimal behaviours.

Recent extensions to the MHP/GOMS framework reveal that not all mental operators are of similar duration as was initially assumed in Card, Moran, and Newell (1983)'s original proposal. Our menu search operator consists of perceptual and cognitive operators. It is an example of a mental operator which is not constant; its value is dependent on where the target item is located in a menu. Also, the designs examined in our study involved a number of different mental retrievals which are assumed to require equivalent duration ($C_M = 1200$ msecs.). A finer grained analysis and better distinctions between proposed designs would be possible if we could differentiate these various mental operators.

In many places in our task analysis, it was difficult to decide whether certain operators were needed because of a lack of a detailed set of guidelines and rules for doing the task analysis. John (1988) points out that it is often difficult to know what the underlying mechanisms are that are driving the internal processes involved in accomplishing a task and whether the analysis successfully explains and predicts performance. This problem is not unique to MHP/GOMS but permeates all forms of cognitive modelling. In general, current cognitive models are guesses of what the underlying mechanisms are. John (1988) suggests that only by producing more successful cognitive modelling cases can we hope to extract the elements of truth about our guesses and to formulate task analysis rules and guidelines. Of course, user interface designers also need adequate training in cognitive modelling in order to generate not only reasonably accurate predictions but also valid ones.

We also need more comprehensive descriptions about MHP operators because not all operators involve a single act of the MHP processors. In fact, many MHP operators (e.g., $P_S(I)$, C_M , $M_{D_M}(I)$, $M_{P_M}(I)$) are composed of several actions of the MHP processors. Aside from the estimates for these operators, the description should provide an operational definition, as in the case of John (1988) (see pp. 44). Otherwise, user interface designers may inadvertently incorporate the same operator twice in their calculations. A case in point is the operator based on Fitts Law. Experiments to determine the estimates associated with Fitts Law (See Eqn(2)) generally involve a series of reciprocal tapping tasks. Each tapping operation delimits the end of one tap and the start of the next tap. Thus, the elapsed

time between two taps inherently incorporates the cognitive and motor operations associated with each tapping task.

Future Work

We took one approach to investigating the issue of user effort. However, because of the emphasis placed by this approach and the simplifying assumptions that were made, there were a number of factors that could not be examined. We discuss possibilities for further analyses.

While task performance times and number and types of MHP operators represent one type of mental-effort measure, mental workload and user errors represent other types. Given the limitations in distinguishing certain mental operations, mental workload may offer a better assessment of mental effort. However, there is very little mental-workload-related work in cognitive modelling (except for [Lerch, Mantei, & Olson, 1989]); the bulk of this research is done in human factors.

The question of which method, history or typing, causes more errors is important as errors generally mean additional effort for error recovery. Because history tools require less physical effort, the potential for making motor errors (typing errors) is reduced, especially for poor typists, but with the increased mental effort, there is more potential for cognitive errors. By the same token, if a memory retrieval in one history-based design requires less effort but tends to be more error-prone (i.e., it retrieves the wrong history item) then this design is not favourable because more overall effort, attributed largely to error recovery, is required. Therefore, it would be informative to examine the issue of user errors but the MHP/GOMS framework does not currently address user errors.

While our analysis did not consider the command generation task, we make some conjectures about how this task influences our results. We conjecture that *typing* would be greatly influenced by command generation. Specifically, users need to expend quite a bit of effort to generate each word in a command and to order the words properly (i.e., according to the Keystroke Level Model, one C_M operator per word in the command). By comparison, we conjecture that users may not need to generate the full command in order to use the proposed history designs. Rather they just need to recognize the command from a menu or generate some part of it so that it can be used as an appropriate criteria (e.g., command name, argument, command number) for retrieving the command. Such recall or recognition operations may take longer than the estimates used in our current study, but certainly not as long as the ones for *typing*. Therefore, if command generation is factored into our user effort analysis, *typing* might perform even worse because of the extra mental retrieval operators. Of course, these conjectures need to be verified by an extended MHP/GOMS analysis.

This study considered the literal reuse of a command which is a natural follow-on to the behavioural study of user recurrences (Chapter 5). It would also be instructive to model and analyze the effort to reuse parts of a previously issued command. However, the main obstacle to such an investigation is the uncertainty surrounding the nature of menu search involving partial matching (i.e., selecting an item that partially matches the search item). As in our literal search case, it is unclear what menu search model is appropriate for partial search or whether the

model for partial search is linear. Thus, an experiment must be conducted to determine an estimate for menu search involving partial match.

Concluding Remarks

This chapter examines the mental and physical effort in a command specification task – the task of specifying a recurrent UNIX command. Two different command specification techniques are examined: retyping the entire command or using a history tool. The extended Model Human Processor and GOMS framework is used to model and predict performance for the typing and history methods. The time to perform cognitive and motor operations for each method and the numbers and types of MHP operators involved are used as measures of mental and physical effort. Three different classes of history-based command specification designs are examined. The history-based designs cover existing as well as hypothetical designs. They also cover designs exploiting a user's recall and/or recognition capabilities.

The main observations from this study are:

- (1) Compared to *typing, command feature* designs favour less physical effort at the expense of more mental effort. They require a range of mental effort (i.e., 1.7 secs to 3.1 secs.); *!name* requires the least mental effort of all history-based designs.
- (2) In *history menu* designs, the *static menu* design requires less physical effort than the *popup menu* because it does not require holding a mouse button down for menu selection. Despite the small overhead costs associated with screen real estate for a menu and switch in focus of attention, the benefits of the *static menu* design are reduced performance time, constant feedback of the contents of a history, and no pauses due to the display of a menu.
- (3) All *step buffer* designs require pauses for system response; the number of pauses depend on the number of items a user examines. If system response times are negligible, pauses are not a deterrent. These designs minimize either physical effort or mental effort.
- (4) Most pure *recall* designs favour less physical effort at the expense of more mental effort. Mental effort is attributed to a memory retrieval of some feature of a desired command (e.g., command number, argument, recency). The design with the least mental effort, *!name*, is the one which retrieves a name from working memory as opposed to long-term memory. Therefore, *recall* designs that require fewer or simpler mental operations as well as fewer physical operations will compete favourably with *typing*, especially when users have average non-secretarial typing skills or better.
- (5) Most pure *recognition* designs have selection techniques that use a different input device. Therefore, there is extra overhead (.7 seconds) associated with homing to different input devices and this may be somewhat annoying. However, fast *recognition* designs are superior to *recall* designs for users with poor typing skills (i.e., less than 40 wpm) because they require less physical effort (1 sec.) and less mental effort (compare 690 + 110/msecs. for menu search to 1200 msecs. for a mental retrieval operation). Furthermore, *recognition* designs provide ancillary benefits such as visual display of a user history.

- (6) The primary operators in the pure *recall* designs are memory retrieval and typing operators. As such, they generally perform better than the pure *recognition* designs when expert and good typists are considered. The bulk of the mental effort in a pure *recognition* design is attributed to visual search; it does not require typing. In relation to the pure *recall* designs, the *recognition* designs using menus are more advantageous for less skilled typists when their constant menu selection times become less significant compared to keystroke times.
- (7) Compared to the pure *recall* designs, a *combined recall and recognition* design is neither superior in mental (2.3 secs.) nor physical effort (6.28 M_K). Compared to the two pure *recognition* designs, the combined design requires less mental effort but more physical effort for users with average or worse typing skills (i.e., less than 55 wpm). In general, the combined design in our study provides no substantial benefits compared to the pure *recall* or pure *recognition* designs. This design requires a large variety of operators: cognitive, perceptual, keystroke and system response operators.
- (8) *Typing* requires less mental effort (1.2 secs.) than all the proposed history-based designs. It favours less mental effort at the expense of more physical effort; it requires the most keystroke operators (10.5 M_K). Therefore, with less skilled typists, *typing* actually incurs a heavy penalty. In fact, it requires the most effort when compared to all history-based designs when non-typists are involved. An average non-secretarial typist (40 wpm) and a typist unfamiliar with a keyboard can specify a recurrent command faster and with less overall effort using a *static menu* design than *typing*.

In summary, while a history-based design for command specification requires more mental effort, it requires less physical effort when compared to *typing*. The additional mental effort is due to memory retrieval of actions. However, if simpler mental retrieval operations are used (e.g., working-memory retrieval or visual search), history-based designs can compete favourably with typing. The fastest *recall* and *recognition* designs, *!name* and *static menu*, are the ones that minimize mental effort and physical effort.

Our study of the accuracy of the predictions generated by the cognitive model using the MHP/GOMS framework revealed that the predictions are well within the goal of 80% of mean observed performance times. We selected one design from each of the three classes of history-based designs and *typing* examined in the user effort study. The mean absolute prediction error of the three history-based designs is 11% while the typing method had an absolute prediction error of 30%. The higher prediction error for the recall-by-argument and typing tasks are attributed to a poor technique for estimating the keystroke operator M_K . Consequently, the predicted times were much higher than the observed times due to an inflated estimate of the keystroke time. In terms of using the existing estimate for the mental retrieval operator, our empirical data indicates that it is a good enough estimate for the operation associated with retrieving the method for executing the history design. Because our estimate of the keystroke operation may be inflated, we were unable to determine the suitability of using the mental retrieval operator in the recall-by-argument task for choosing a distinctive command argument as a selection pattern. However, the absolute prediction error of this task is 19% and this is within the 20% error goal.

Chapter 7

Conclusion

The idea of allowing users to make use of data and actions from an earlier part of their interactions within the context of their current interactions is the basis of many history-based tools that system designers have provided. However, these tools are not an integral part of any user-support tool suite. Furthermore, there is little empirical evidence that such history tools are effective and little information concerning the important characteristics of a history-based, user support tool. Therefore, a natural research question is whether history tools can, and do, enhance and support user interactions?

This question is examined from three different perspectives: design, user behaviour, and human information processing. Our first set of studies, in Chapters 2 and 3, examined history-tool artifacts to identify seven history uses, to assess the degree to which current artifacts support all seven history uses, and to propose requirements and options for enhancing current designs. Our second set of studies, in Chapters 4 and 5, examined real user behaviours for information about command recurrences and history-tool usage. Our final set of studies, in Chapter 6, examined the mental and physical effort associated with issuing a recurrent command and the implications that mental and physical effort have on the design and use of history tools.

Except for the first set of studies, the primary focus is on history tools which support the literal reuse of previous interactions. Other history uses and other ways to reuse activities (e.g., partial reuse) have not been examined in detail. However, based on our studies of reuse, the results are favourable towards the use of history tools for supporting recurrent user interactions.

Contributions

There are four major contributions of this thesis. The first is that our findings, along with Greenberg (1988b)'s, represent an accumulating collection of information about the behavioural, psychological, technological, and user-task factors influencing the design and use of history tools. Our studies represent only one of two formal examinations of history tools and the only ones to have examined the cognitive and behavioural merits and drawbacks of history tools for user support. Our primary findings are as follows:

- (1) We uncovered seven potential uses of history (i.e., reuse, inter-referential I/O, error recovery, navigation, reminding, user modelling, and user interface adaptation).
- (2) We observed that current history tools do not support all seven history uses and we proposed requirements and options for history tools that integrate all seven history uses.
- (3) We observed that *cs*h and *tc*sh users make limited and simplistic use of history tools.
- (4) We observed that users make repeated references to a small group of commands at certain intervals in their session. We demonstrated that this user behaviour, locality of command recurrences, is a more meaningful characterization of command recurrence behaviour both in terms of accounting for history use and predicting reuse opportunities and reuse candidates.

Locality and working set were examined previously within the context of window references [Henderson & Card, 1986a; Card, Pavel, & Farrell, 1984], but our study of locality replicated these earlier observations for a finer grain of behaviour: issuing command directives to a system. This replication and generality is an integral part of any research, and especially research in HCI. More importantly, our research demonstrated, in a formal way, that locality in user commands is not a randomly occurring behaviour but a behaviour which arises as a direct result of user-computer interactions.

The Greenberg (1988b) study provided the initial observation that users repeat their commands frequently, especially recently-issued commands. Our studies followed up on his study by corroborating his findings and demonstrating that locality is a better characterization of command recurrence. Greenberg (1988b) also provided insights into the performance of different history-prediction techniques that exploit this recency characteristic. We proposed an alternative which takes locality into account.

- (5) We used cognitive modelling to generate predictions about the mental and physical effort involved in the task of specifying a recurrent command for a given method and design. Thus, we do not model the task of deciding which method (i.e., history tool or typing) or which history-tool designs to use for command specification. We assumed that the command had been generated and appeared in the user's working memory. We also assumed that users exhibit expert, error-free task performance behaviour. Within the scope of these constraints, we drew three conclusions. First, most of our proposed history tool designs favour less physical effort at the expense of more mental effort while re-typing favours less mental effort at the expense of more physical effort. Second, the increased mental effort associated with using history tools can be alleviated by designing history tools that exploit simpler mental operations (e.g., working memory retrievals and perceptual processing). For example, memory retrieval could be limited to working memory and perceptual processes could be used rather than long-term memory retrievals. Third, despite the penalty associated with switching from typing to history tools, non-expert typists using history tools do expend less overall effort.

The second thesis contribution is the demonstration of the value of using different research methods to study the impact that history tools have on user

interactions. Design is a complex endeavour and requires input from many different perspectives including user task, technology, user behaviour, and human information processing. These various perspectives offer insights into task requirements, technological barriers, behavioural factors, and human information processing constraints. Furthermore, it is important, in any investigation of factors influencing the use of a tool, to involve different granularities of analysis because this can provide macroscopic and microscopic perspectives on important problems and issues which otherwise would not be evident. This thesis is unique in that it examined the prospects of history-based user support from three different perspectives and grains of analysis (i.e., design, user behaviour, and human information processing), demonstrating how the various studies can provide insights into the design problem (i.e., history-based user support tools).

The third thesis contribution relates to our findings regarding locality in command recurrences. These findings provide insights not only for the design of history-based user support tools but also for the design of user interfaces in general. In particular, locality is a basic behavioural phenomenon which user interfaces need to consider and address. Our study revealed poor locality in command recurrences and this observation leads to two interesting conjectures that merit further investigation. One conjecture is that current interfaces do not provide adequate tools and resources so that users can keep unrelated work in separate workspaces. As a result, a user history is a collection of different threads of user activities rather than a single thread of user activity. A second conjecture is that poor locality is an indication of a good user interface design. A good user interface design imposes fewer constraints on users and as a result, users are better able to accomplish their tasks with less trivia because of user interface flexibilities. Also, this means that users are not forced to use the same method to accomplish their tasks; they can use different variations. This variation would be manifested in the form of less locality in user interactions. A study into the cause of poor locality is required before we can conclusively explain the observation of poor locality in user interactions.

The final contribution of this thesis is the contributions made in the area of cognitive modelling. These contributions include:

- (1) An estimate for a new Model Human Processor operator (i.e., visual search of a menu whose items are organized in the order in which they were last referenced starting with the most recently referenced menu item),
- (2) A demonstration of the ability of MHP/GOMS to model user activities (e.g., visual search of a history menu with a known organization, menu selection, and associative memory retrieval) within a new and different task domain (i.e., history tools), and
- (3) Additional corroboration that the new and revised estimates of MHP operators (i.e., menu search and mouse selections) are able to model and predict task performance times to within 80% of mean observed times.

Future Directions

There are four research directions in which to extend the work begun in this dissertation: studies of other aspects of history for reuse, studies of other

behavioural patterns motivating uses of history tools, studies of the cognitive and design issues influencing history-based, user support tools, and enhancements to cognitive modelling.

First, the dissertation examined the literal reuse of previous user interactions. However, there is also the issue of partial reuse of previous user interactions which needs to be examined. This includes how people reuse past interactions, which elements in past interactions are most useful, and the mental and physical effort associated with partial reuse of previous user interactions.

Second, behavioural studies are important for identifying and characterizing behavioural patterns evident in user-computer interactions. As demonstrated in this thesis, such behavioural studies provide important insights into the support that users need in their interactions with the computer. Further research is needed a) to identify and characterize behavioural patterns that influence other uses of history tools like the ones identified in Chapter 2 (e.g., navigation, error recovery), b) to examine the implications that each behavioural pattern has on the design of history tools in a fashion similar to that described in Chapter 5 and in Greenberg (1988b), and c) to study the impact that computing artifacts have on locality (e.g., tasks which have more or less locality than other tasks) so as to shed light on the design of artifacts.

Third, research is needed in the design of history tools. An important conclusion from this dissertation is that current history tools are woefully inadequate in regard to cognitive considerations influencing their use. Research is needed to identify important cognitive issues influencing the use of history tools and to formulate solutions to address these issues. A starting point for this research effort is to extend the design issues and requirements enumerated in Chapter 3 and to explore the design solutions outlined therein.

Finally, psychological model building allows designers to explore their design intuitions, to explore and understand the mental and physical processes and constraints underlying the use of a design, and to predict the efficacy of design proposals. Cognitive modelling shows promise not only in terms of providing a bridge between psychological theories about human behaviour and design proposals for supporting them but also in terms of influencing its own development as a research tool. That is, by using the methodology to explore real design issues, we can identify its limitations. Therefore, we can help drive efforts to broaden the methodology's scope, applicability, and usefulness.

Appendix A

Selected Systems Supporting History Tools

Six systems were selected for assessment: ALOE, INTERLISP-D, MINIT, MPW, ROOMS, and SEED. In order to provide a fair assessment of the state of current support for all uses of history, only general-purpose development environments were considered rather than special-purpose systems (e.g., text editors, information retrieval systems). Such environments support computing activities by providing access to different applications (e.g., text editing, graphical drawing, electronic communication, programming, and managing file systems).

ALOE is an interactive structure-editor-based generator for programming environments [Linxi & Habermann, 1986]. ALOE, and the environments generated by ALOE, support a history tool which records program development history. The history tool provides facilities that allow users to reuse command lines and recover from errors. Command lines may be reused individually or as a group, facilitated by a history macro facility. User errors may be corrected using an editor or using an UNDO/REDO facility.

INTERLISP-D is a display-oriented programming environment supporting the LISP programming language and a set of user facilities [Teitelman & Masinter, 1981; Teitelman, 1977]. DWIM and PROGRAMMER'S ASSISTANT are two notable user facilities that support history tools. In certain spelling error situations (e.g., function name), DWIM's spelling corrector scans related items the user recently worked with (e.g., functions) to make corrections. PROGRAMMER'S ASSISTANT maintains a history list containing user operations, a description of their side effects, and their results. As well, several commands for manipulating a history list are provided: REDO one or more operations, UNDO effects of specified operations, FIX operations before re-executing them, and USE a substitution before re-executing the operation.

MINIT allows users to issue a single textual command line combining command submission and window management operations [Barnes & Bovey, 1986]. It exists on a multi-window, single-user, graphical workstation running the UNIX operating system. As part of its support for command submission, MINIT maintains a history menu containing entered commands. The history tool allows users to reuse previous commands with possible modifications.

MPW – MACINTOSH PROGRAMMER'S WORKBENCH – is the MACINTOSH development environment [Farr, 1989]. The MPW environment provides an array of application

creation tools along with a command language called MPW SHELL. MPW SHELL combines the features of a command shell and a multi-window text editor with mouse-based command capabilities. It supports operations typically found in most UNIX command interpreters (e.g., file management and input and output redirection). Users may use the MPW “Worksheet” to execute and store often-used commands; the current activity is noted at the bottom of the “Worksheet”. Furthermore, users may re-execute commands in any window. For example, users may re-execute commands embedded in source code windows, in program comments, in program output, or in the “Worksheet”. Commands are executed at the location where they are invoked and outputs appear in the area following each command. If commands are re-invoked, previous outputs are replaced with current outputs. This manner of command execution makes it difficult to reconstruct the order in which commands are executed because there is no semblance of a linear history.

ROOMS is an enhancement of INTERLISP-D that supports multiple virtual workspaces. Each room represents a virtual workspace consisting of all programs used to perform a single user task and no two rooms are displayed simultaneously. In addition to the history capabilities available in INTERLISP-D, ROOMS provides history capabilities for navigating around multiple workspaces. Users can determine where they are by examining the display. BACK DOOR allows users to go back to the previous activity space and find out where they just came from.

SEED is a session editor implemented on top of the VAX/VMS environment [Holsti, 1989]. In addition to the traditional means of command execution (one command at a time), SEED supports incremental script creation and script processing. A transcript of all executed and unexecuted commands for a session, including user mistakes, recovery actions, and command outputs, may be saved and restored from session to session. A transcript’s contents may be copied and manipulated before it is incrementally re-executed. Modifications to executed and unexecuted commands are permitted. However, executed commands are automatically undone if they are modified (i.e., the system backtracks to the state preceding the modified command before it is re-executed). Therefore, the transcript provides a linear record of the history.

Appendix B

Locality Detection Algorithm

As a program executes, it makes references to portions of a computer's memory in units known as *segments*. The sequence of memory references is a *memory reference string*. The set of segments referenced during a program's execution is the *memory reference set*.

Locality is the phenomenon in which a program's memory references are limited to a small subset of segments for an extended time interval. The intuitive notion of a *locality* has two components: a favoured subset of segments (i.e., a *locality set*) and a definite reference interval (i.e., a *phase*) over which this favoured subset remains unchanged.

A least recently used (LRU) stack may be used to construct sets of segments which possess many of the characteristics associated with a locality. The LRU stack can be represented as an ordered vector $L(t) = (L_1(t), L_2(t), \dots, L_n(t))$ where n is the size of the memory reference set and $L_i(t)$ is the segment identifier for the i th most-recently-referenced segment at time t . If the stack is cut at any position i , then the topmost i segments in the stack $S_i(t) = \{L_1(t), L_2(t), \dots, L_i(t)\}$ are the i most recently referenced segments at time t . Since the LRU stack can be cut at any point, it defines a hierarchy of localities (i.e., $i = 1, 2, 3, \dots, n$).

However, the problem with using the LRU stack is that, at each point in time, the number of locality sets is equal to the number of segments in the memory reference set (i.e., n). The reason is that the LRU stack does not contain enough information to allow the selection of certain locality sets as being more "distinctive" than others. In order to do this, we need information about the formation time (i.e., $F_i(t)$) and termination time (i.e., $T_i(t)$) of a set of segments $S_i(t)$. Also, we need to maintain the most recent time unit $R_i(t)$ where a reference was made to the i th stack position (i.e., last time where a reference was made to the least recently referenced member of the set of segments).

The formation time of a new set of segments of size i is the termination time of the previous set of segments of size i . Thus, a set of segments of size i at time t , $S_i(t)$, remains intact until a reference is made to a stack position greater than i . It is considered distinctive (i.e., a locality set) if and only if every member of the set has been re-referenced since the set was formed. Specifically, a locality set at time t , $LS_i(t)$, is any $S_i(t)$ for which $R_i(t) > F_i(t)$.

In order to detect such locality sets, an extended LRU stack is needed [Madison & Batson, 1976]. It consists of the LRU stack and two ordered vectors:

$$F(t) = (F_1(t), F_2(t), \dots, F_n(t))$$

$$R(t) = (R_1(t), R_2(t), \dots, R_n(t))$$

There are two parts to the locality detection algorithm. In the first part, an active locality set $LS_i(f)$ is terminated at time $t = e$ when $f \neq F_i(e)$. We must check for termination because for a given size i , the termination of a locality set is brought on by the activation of a new set of segments for the same size i . In the second part, a new locality set is formed at time $t \neq f$ when $R_i(f) > F_i(f)$ (i.e., at time $R_i(f)$ when the least recently referenced segment is re-referenced after the locality set is formed at time $F_i(f)$).

The following is an outline of the locality detection algorithm developed from Madison and Batson (1976).

Assumption: Each reference constitutes one time unit

Given:

- a two dimensional array LRU for the LRU stack
(i.e., $LRU[i, t]$ holds the i th stack element at time t)
- a two dimensional array F for the formation time
(i.e., $F[i, t]$ is the formation time of a $S_i(t)$ at time t)
- a two dimensional array R
(i.e., $R[i, t]$ is the most recent reference to i th reference at time t)
- variable n is the length of the current reference set
- variable $reference$ is the current reference
- an array $Active$ numbered from 1 to n , holds a boolean indicating whether S_i is an active locality set
- an array $FormTime$, numbered from 1 to n , holds the formation time of LS_i
- a two dimensional array $Sets$ holds the references in the current active locality set
(i.e., $Sets[size, i]$ holds the i th member of locality set of size $size$)
- $whereInLRU(argument)$ is a function returning position of the $argument$ in LRU at time $t-1$

```

/* process each reference and see if it is part of a locality set */
t = 1
while (true) {
    reference = readReference()
    if (reference = null)
        exit()

    lastOccur = whereInLRU(reference)
    if (lastOccur <= n) {
        /* the reference appeared previously in LRU[lastOccur, t-1] */
        LRU[1, t] = LRU[lastOccur, t-1]
        R[1, t] = t
    }
}

```



```

F[lastOccur, t] = F[lastOccur, t-1]

for (j=1 to lastOccur-1) {
    LRU[j+1, t] = LRU[j, t-1]
    R[j+1, t] = R[j, t-1]
    F[j, t] = t
}
for (k=lastOccur+1 to n) {
    LRU[k, t] = LRU[k, t-1]
    R[k, t] = R[k, t-1]
    F[k, t] = F[k, t-1]
}
}
else {
    /* the reference is a new reference */
    LRU[1, t] = reference
    R[1, t] = t
    F[1, t] = t
    if (t > 1)
        for (j=1 to n) {
            LRU[j+1, t] = LRU[j, t-1]
            R[j+1, t] = R[j, t-1]
            F[j+1, t] = t
        }
    n = n + 1
    Active[n] = 0
}

if (t > 1)
    for (i=1 to n) {
        /* Part 1: Look for terminated locality sets */
        if (Active[i] = 1 && FormTime[i] ≠ F[i,t]) {
            /*  $LS_i(t-1) = \{LRU[1, t-1], \dots, LRU[i, t-1]\}$ 
            /* print  $LS_i(t-1)$ , formation and termination times */
            print FormTime[i], t-1
            for(m = 1 to i)
                print Sets[i, m]
            Active[i] = 0
        }

        /* Part 2: Look for new locality sets */
        if (R[i,t] > F[i,t] && Active[i] = 0) {
            Active[i] = 1
            for (j=1; j<=i; j++)
                Sets[i, j] = LRU[j, t]
            FormTime[i] = F[i, t]
        }
    }

```

```
        }
    t = t + 1
}

for (i=1 to n)
    if (Active[i] = 1) {
        /*  $LS_i(t-1) = \{LRU[1, t-1], \dots, LRU[i, t-1]\}$  */
        /* print  $LS_i(t-1)$ , formation and termination times */
        print form[s], t-1
        for(m = 1 to i)
            print Sets[i, m]
    }
```

Appendix C

Cognitive Modelling Framework

The cognitive modelling analysis framework used in this thesis consists of the Model Human Processor (MHP for short) and a representation and methodology, known as GOMS, for describing a user's task.

Model Human Processor

Model Human Processor is a general characterization of human information processing (Figure 1); it includes a system architecture and a set of quantitative parameters of component human performance. The system architecture is made up of three interacting subsystems – perceptual, cognitive, and motor – together with a set of ten principles of operation¹ derived from psychological evidence and a human information-processing model of psychological behaviour. Each subsystem has its own memories and processors. The principles of operation describe a wide range of operations and control of these memories and processors.

Each processor operates serially but concurrently with other processors, subject to limitations imposed by data flow requirements. The perceptual processor receives information from the outside world and deposits it into the appropriate sensory memory – visual image store or auditory image store – where it is held until it is symbolically encoded. Working memory receives encoded information which is then used by the cognitive processor, along with previously stored information from long-term memory, to make decisions about what to do. The cognitive processor deposits information into working memory which, in turn, initiates the associated motor processor actions that act on the outside world.

There are quantitative parameters associated with each subsystem component. A memory component has three parameters: storage capacity μ , decay time of an item in the memory δ , and main code type (physical, acoustic, visual, or semantic) κ . A processor component has a cycle time τ parameter – time to process one unit of information. In the original model, a parameter value may be expressed in terms of a *typical* value or as a lower and upper bound (i.e., *range*).

¹ Rationality Principle: People attain their goals in a rational manner, given a task structure and inputs, bounded by limitations on their knowledge and processing ability.

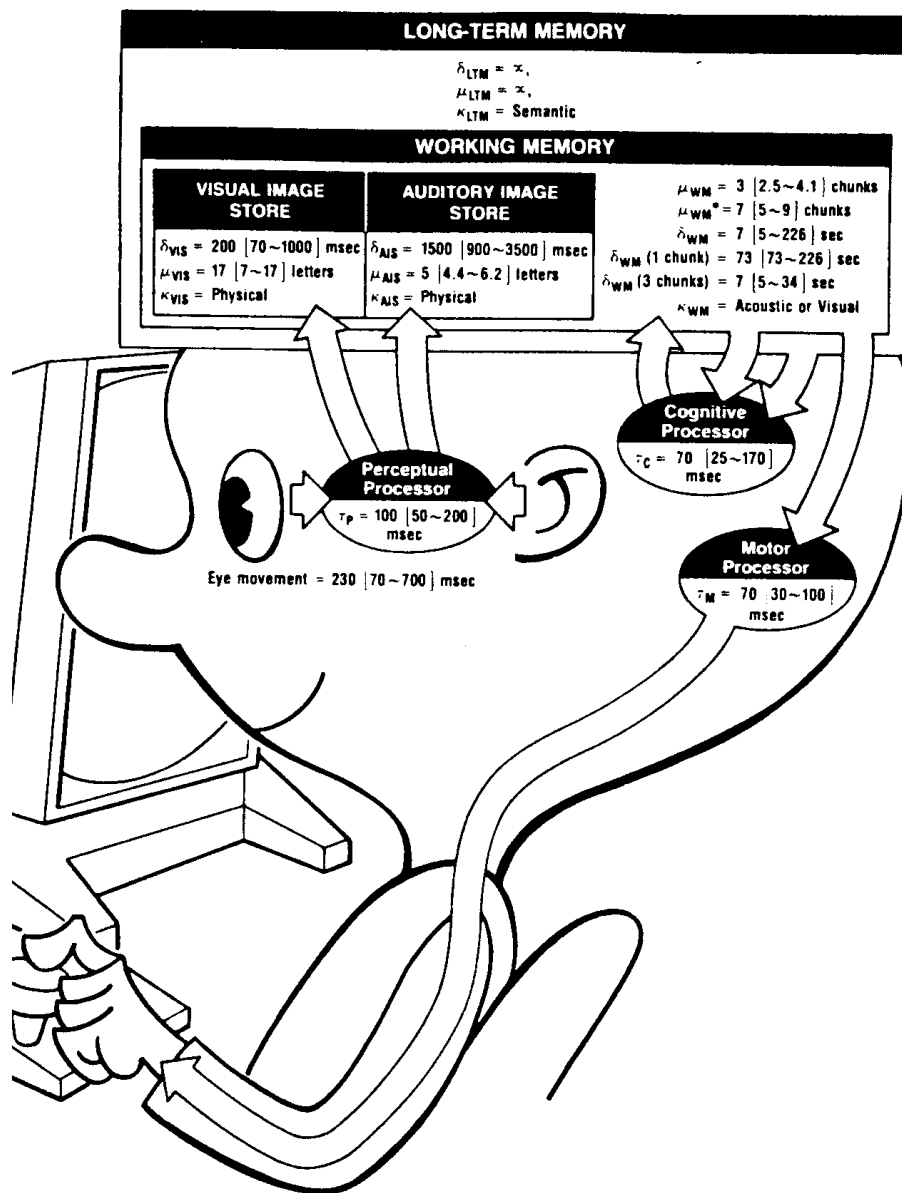


Figure 1: The Model Human Processor – memories and processors [Card, Moran, & Newell, 1983, (pp. 26)]. Sensory information flows into Working Memory through the Perceptual Processor. Working Memory consists of activated chunks in Long-Term Memory. The basic principle of operation of the Model Human Processor is the *Recognize-Act Cycle of the Cognitive Process*. The Motor Processor is set in motion through activation of chunks in Working Memory.

Elemental acts of the processors are known as *operators*. A task analysis in this framework involves subdividing a task into the gross functions performed by each processor and decomposing these functions into sequences of operators. Approximate engineering-style calculations of the times to carry out gross-level human behaviours are determined from task descriptions [Card, Moran, & Newell, 1983]. MHP assumes that operators are independent of the task environment so that they can be combined to perform many different tasks. It also assumes that operators act independently of each other so that the time to perform a sequence of operators is the sum of the times to perform each operator. Example 1 illustrates a sample task from Card, Moran, and Newell (1983) which shows how a task is modelled and a calculation is made.

GOMS

GOMS is a representation and methodology for describing what a user needs to know in order to perform computer tasks (i.e., a user's cognitive structure). User knowledge is represented in terms of four cognitive components of skilled task performance:

- (1) Goals are defined in terms of user tasks to be performed.
- (2) Operators are elementary acts of the three processors which result in changes to a user's mental state or to an external state (e.g., pressing a key).
- (3) Methods, associated with goals, are procedures for accomplishing a task and are encoded in terms of the operators.
- (4) Selection rules are used to choose amongst a number of methods for achieving a particular goal.

A family of engineering-style cognitive models, based on MHP and formulated at different levels of GOMS analysis, has been developed for a number of complex HCI task domains. The Keystroke-Level and Unit-Task Level Models are two such models formulated at the keystroke and unit-task level, respectively [Card, Moran, & Newell, 1983]. Each model predicts the amount of time required by a user to do representative tasks in the modelled task domain.

The Keystroke-Level Model is based on a simplified GOMS analysis which does not require an explicit analysis of goals and selection rules. It is used in situations where it is possible to specify a user's interaction sequence in detail. However, it has two additional restrictions: a task method be given in order for time predictions to be made and predictions apply only to the execution time of a task. This model has been empirically validated and tested for 14 tasks (i.e., 4 editing, 5 graphics, and 5 command executive), 11 different systems (i.e., 3 editors, 3 graphics systems, and 5 command executive subsystems), and 28 users [Card, Moran, & Newell, 1980].

Critical Elements of a Cognitive Model

In order for designers to use such cognitive models, there are a number of critical elements they must possess: calculation, approximation, zero-parameter prediction, learnability and usability by computer designers, and coverage of the

Example 1: A simple reaction-time analysis using the Model Human Processor [Card, Moran, & Newell, 1983, (pp. 66-69)]. A user sits before a computer display. Whenever any symbol appears, the user responds by pressing the space bar. What is the time between signal and response?

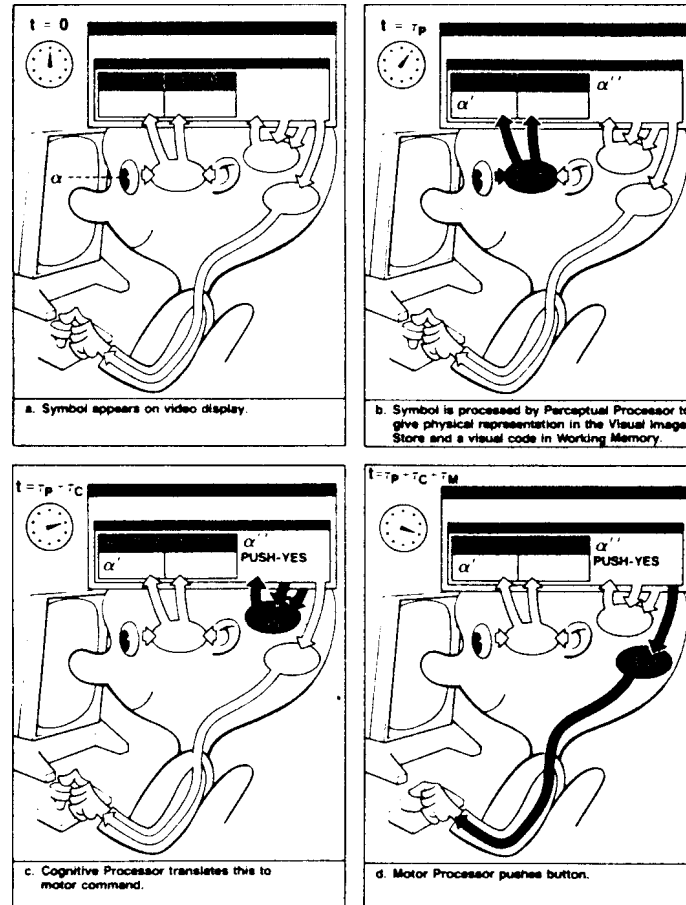


Figure 2: A reaction time analysis using the Model Human Processor.

Solution: Figure 2 illustrates the course of processing through the Model Human Processor. The user is in some state of attention to the display (Figure 2a). When some physical depiction of the letter A – denoted by α – appears, it is processed by the Perceptual Processor, giving rise to a physically-coded representation of the symbol – written as α' – in the Visual Image Store and very shortly thereafter to a visually coded symbol – written as α'' – in the Working Memory (see Figure 2b). This process requires one Perceptual Processor cycle τ_P . The occurrence of the stimulus is connected with a response (Figure 2c), requiring one Cognitive Processor cycle, τ_C . The motor system then carries out the actual physical movement to push the the key (Figure 2d), requiring one Motor Processor cycle, τ_M . Hence, the total time required between signal and response is $\tau_P + \tau_C + \tau_M$.

total task [Card, Moran, & Newell, 1983; John, 1988]. Calculation supports predictions based on explicit mathematical models of user behaviour with the task as opposed to judgements, designer's experiences, or evaluations. Approximation is the recognition that the processes involved in user behaviour are extremely complex and that real-world calculations for engineering purposes need only include sufficient detail to do the design job. Zero-parameter prediction stipulates that parameters of the engineering models used to make predictions be determined during construction of the model and prior to its application to a new situation. The basic psychology must be built into the model so that the model can be taught to and be usable by computer-system designers who are not trained psychologists or human factors experts. Finally, an engineering model of a specific task domain must provide coverage for the total task by accounting for all first-order effects observed in the task domain.

Shortcomings of Engineering Models

In the initial effort demonstrating how the proposal would be realized, a number of simplifying assumptions were adopted. Some of these assumptions represent shortcomings of current engineering models which are enumerated herein [Card, Moran, & Newell, 1983; Olson & Olson, 1990]:

- The initial model accounts for skilled and error-free user behaviour but it does not consider unskilled user behaviour and the effects of errors, learning and casual usage on user performance.
- The initial model focuses on the usability of a system and does not address the functionality of a system (i.e., what computer tools are needed).
- The initial model uses a mental operator to represent all cognitive processes instead of a differentiated treatment of such cognitive processes which take different amounts of time.
- The initial model assumes that the human information processing components operate in a serial fashion and provides no account of the parallel processes involved in user behaviour.
- The initial model does not address the issue of how much information is held in memory when a computer system is used (i.e., mental workload), of individual differences amongst users, of user fatigue, and of the social and organizational considerations associated with computer-supported work.
- The model does not provide guidance in predicting the usefulness, user satisfaction, and user acceptability of a system.

Recent cognitive modelling efforts are attempting to address a number of these shortcomings (see Olson and Olson (1990) for a review of research in this area) but the work is still very much in its infancy.

Extensions to MHP and GOMS

A number of researchers have extended Card, Moran, and Newell (1983)'s work along two fronts. First, they have confirmed the basic set of parameters of the processors in MHP and extended the number of parameters to include the time to perform component activities for a number of new HCI task domains [Olson & Olson, 1990]. Second, John (1988) has developed two additional GOMS models by

extending MHP and GOMS analysis to include two new HCI task domains: stimulus-response compatibility and transcription typing.

Both of these developments have direct implications for the studies in Chapter 6 and Appendix D. MHP's parameters have been extended to include task activities that are particularly relevant to those that underlie the task of using a history tool. In particular, John, Rosenbloom, and Newell (1985) and John (1988) extended MHP to include a class of user behaviour involving operators that are at a lower level than any operators found in previous GOMS models. Furthermore, John (1988)'s work demonstrated how to describe such user behaviours using an algorithmic GOMS formulation. This formulation is adopted as the representation for task descriptions used in this thesis. The following section is a brief presentation of the algorithmic GOMS formulation used in John (1988).

GOMS Model of Immediate Behaviour

Stimulus-response compatibility (SRC for short) is a robust psychological phenomenon where the degree of difficulty of a response is dependent on the complexity of the relationship between the set of stimuli and their corresponding responses [John, 1988]. It was selected as the domain within a class of user behaviour, known as *immediate behaviour*, in which to expand GOMS. Immediate behaviour is the direct mapping of a stimulus into a response without problem-solving or planning [John, 1988]. Such behaviours are examples of routine cognitive skill and are generally associated with reaction time tasks where the response to a stimulus is well known and initiated immediately upon presentation of the stimulus, without deliberation about what response is appropriate.

ARTLESS: An Algorithmic Response-Time Language

Rosenbloom suggested that people perform SRC-related tasks by executing algorithmic programs and proposed an informal, algorithmic response-time language, ARTLESS, for describing the performance of such tasks [John, 1988]. This language was used by John (1988) to describe SRC and transcription typing tasks. A longer description of the language appears in John (1988).

The features of the language are:

- (1) Tasks are accomplished with a sequence of MHP processor operations.
- (2) Each operation has an identifiable duration that is only dependent on the processor performing the operation.
- (3) Tests in conditionals must be exhaustive and explicit because there is no IF-THEN-ELSE construct (i.e., two ARTLESS IF-THEN statements are used instead – testing for the conditional and its negation).
- (4) Tests to exit a loop take time only when they succeed.
- (5) Reasonable algorithms are purely functional (i.e., there are no null operations or empty loops).

In subsequent descriptions of ARTLESS, an example task description using ARTLESS (Example 2), drawn from our history effort analysis, is used to illustrate the language's constructs. ARTLESS has seven constructs: constants, variables,

Example 2: In the *cs*h history tool, a previous command line containing an *argument* is reused by the history command *!?argument*. As a simplification, we will assume that the history tool looks for the most recent occurrence of the command line in the history containing the given *argument*. Furthermore, let us assume that the average length of a typical UNIX command argument is \bar{n}_A characters long. Therefore, an ARTLESS description of the cognitive and motor processes involved in issuing the history command *!?argument* is as follows:

```

L1   CM   Get-!? (<!>)
L2   CE   Initiate (<!>)
L3   MK   Key (<!>)
L4   CE   Initiate (<?>)
L5   MK   Key (<?>)
L6   CM   Get-One-Argument ("Argument")
L7   Count ← 1
L8   REPEAT BEGIN
L9   CE       Initiate (Argument[Count])
L10  MK       Key (Argument[Count])
L11  Count ← Count + 1
L12  UNTIL Is-Equal? (Count,  $\bar{n}_A$ )
L13  CE   IF-SUCCEEDED Is-Equal? (Count,  $\bar{n}_A$ ) THEN BEGIN
L14  CE       Initiate ()
L15  MK       Key ()
L16  END

```

assignments, blocks, operators, branches, and loops. Constants, variables and assignments are like typical programming language constructs (e.g., 1, Count, Count ← Count + 1). Blocks – beginning with the keyword BEGIN and ending with the keyword END – delineate sequences of operations (see lines L13 - L16).

Operators are parameterized black boxes that do the real work in the system (e.g., Get-!? (<!>) retrieves the method for the !?argument task). Lines L1-L6, L9-L10, and L12-L15 contain operators. Predicates are a subclass of operators that don't return a value but are used to test for the truth of some condition (e.g., Is-Equal? (Count, \bar{n}_A) test for the truth of equality of the two arguments – Count and \bar{n}_A). Lines L12 and L13 contain predicates. Algorithms are specified without the need to detail how the concept "Argument" is represented.

Branches permit the testing of a predicate and the subsequent execution of an operator or a block of operators when the predicate is true. Lines L14 and L15 are executed only if the variable Count and the constant \bar{n}_A are equal (see L13). Since, an ELSE clause is not explicitly supported, a branch requiring such a clause may be obtained by an explicit test for the complement of the IF clause.

Finally, a loop executes an operator, or block of operators, until some predicate becomes true (see Lines L8 - L12). There must be an explicit test for the predicate at the end of the loop – line L12 is the explicit test for the REPEAT loop in line L8 while the predicate test in L13 is provided for readability.

Cost Scheme for an ARTLESS Algorithm

A simple scheme is used to assess the cost of an algorithm.

- (1) Except for operators, all constructs in ARTLESS are free of charge. In Example 2, lines L7, L8, L11, L12, and L16 are free because they do not involve the operator construct.
- (2) Each operator takes one unit of time associated with the processor that performs the operation. The operator times are expected to relate to the respective processor cycle times for the MHP. Therefore, lines L2, L4, L9, L13, and L14 involve the cognitive processor (note the C_E adjacent to the lines), while lines L3, L5, L10, and L15 involve the motor processor (note the M_K adjacent to the lines). Both lines L1 and L6 involve operations that are not elemental operations of the cognitive processor and are each assigned a composite cost of C_M . Line L1 requires a retrieval of the history command (i.e., $!?$) from Long Term Memory while line L6 selects one of the arguments in the desired command line as the pattern to the history command.
- (3) The REPEAT loop is a special case for counting operators. All operators within the loop count as they are executed. The predicate in the UNTIL line of the loop does not count, but the predicate outside the loop – which retests the termination of the loop – does count. This is why line L12 has no cost assigned to it but the predicate test in L13 does, to represent a one time cost for the REPEAT loop in lines L8 - L12.



Appendix D

Visual Search of a History Menu

This appendix presents an experiment to estimate a MHP operator required for our analysis of the user effort required to issue a recurrent user operation (see Chapter 6). The reason for the estimation experiment is that the MHP/GOMS framework does not model visual search of a menu and is therefore unable to predict user behaviour involving this task. Furthermore, this information is not available in a form which can be incorporated into the framework. The reason is that there are conflicting theories about the visual search strategy that people use.

Estimation of Visual Search Operator for a History Menu

Studies in cognitive psychology have shown that when a sequence of alphanumeric characters is displayed in the foveal (eye focus) region of the eye (eliminating the need for eye movement), an exhaustive search strategy is used (i.e., each element is examined). On the other hand, when a large visual list is involved, necessitating eye movements, a self-terminating search strategy is observed (i.e., search terminates once the search item is found) [Chase, 1986]. Typically, a menu is sufficiently large to necessitate eye movements so that a self-terminating search strategy would be in effect.

On the other hand, studies of menu search within HCI using real-world stimuli have suggested that knowledge (e.g., menu organization and experience) plays a role in directing and limiting search of menus [Lee & Raymond, to appear; Mehlenbacher, Duffy, & Palmer, 1989]. Specifically, users can narrow the search space to a smaller area using knowledge about the menu. As a result, menu parameters such as organization, frequency of reference, match criteria (e.g., literal, categorical, partial) could play a role in narrowing the search space and influence the visual search strategy used [Giroux & Belleau, 1986; Paap & Roske-Hofstrand, 1988].

Current menu search studies have suggested two candidate models: a systematic self-terminating search and a random search [Card, 1984; MacGregor & Lee, 1987a]. In a systematic self-terminating search, users use some strategy (e.g., top down) to keep from examining an item that has been previously examined and users terminate the search process as soon as the examined item matches the search item (i.e., target item). This search strategy is based on empirical studies of search times for categorical information (i.e., search that

involves matching a target with its appropriate category). In contrast, items are examined randomly in a random search, some having been examined previously, until the target item is located. This search strategy is based on empirical studies of search times and eye movement data for visual search of command menus organized in one of three ways: function, alphabetic, and random. There is currently some controversy as to whether the current data differentiate between the two models [Card, 1984; MacGregor & Lee, 1987b]. Aside from differences in strategies, the two models would also predict different search times.

Given the lack of a unified visual search theory as well as a lack of a clearcut visual search strategy for our situation, we ran an experiment to determine the strategy for our history-menu search task. There were two objectives for this experiment. The first objective was to determine the strategy used in a visual search of a history menu. The second objective was to estimate the time to perform this task for the menu search operator.

Method

Our experiment is modelled closely after Neisser (1964)'s visual search paradigm. It is two independent experiments conducted together. Each experiment is a two-factor within-subjects design measuring visual search performance with a history menu. However, menu references exhibited *locality* in the first experiment and *non-locality* in the second experiment. Search items were always present on the menu in the locality experiment and were either present or absent on the menu in the nonlocality experiment.

Serial position is a factor which designates where a search item appears in the menu. Serial position 1 is the top item in the menu and serial position 8 is the bottom item in the menu. *Scenario* is a factor which emulates either the locality or non-locality menu reference behaviour using different constraints and probability distributions. The probability distributions represent the likelihood of finding a search item at each menu position. Two different scenarios are considered in each experiment.

Subjects

16 paid volunteers, consisting of students and staff at the University of Toronto, participated in the experiment. All subjects had knowledge and experience with UNIX. Subjects served in both experiments in one session. This is a deliberate attempt to mirror a typical user session in which their interactions exhibit both locality and non-locality behaviours (see Chapter 5).

In each half of the session, subjects performed one of the scenario conditions from each of the two experiments. No two scenario conditions from the same experiment were administered in the same half of the session. Given this restriction, there were 16 possible combinations for the administration of the scenario conditions. The administration of the scenario conditions were counterbalanced to eliminate possible biases resulting from sequencing effects by randomly assigning a subject to one of the 16 possible combinations.

Materials and Apparatus

A list of 12 items was used with the 8 most recently referenced items being visible on the menu (i.e., the top eight items in the list are visible while the bottom four items in the list are never displayed). The term *menu* is used to refer to the visible portions of the list. The most recently referenced item appears at the top of the menu and the 8th most recently referenced item appears at the bottom of the menu.

The stimuli are UNIX command lines. Table 1 lists the menu items used for each scenario factor in the two experiments. All the items are of the same length because our primary objective is to study how locality and non-locality effects influence visual search in the absence of other influencing factors.

The experiment machine is a SUN 3/50 workstation consisting of a high-resolution graphic display and keyboard. A simple window manager MGR from Bellcore is used. The SUN display is divided into three windows (see right screen panel of Figure 1). The top window contains instructions to the subjects. The bottom window displays an error message whenever the subject makes a mistake (e.g., selecting an inappropriate key on the keyboard). The middle window is the primary interaction window where a list item, the menu, or a search item is presented and subject responses are made.

Scenarios in Locality and Non-Locality Experiment

In the locality experiment, two scenarios were examined: *small locality set* and *large locality set*. Search items in the small locality set scenario appear primarily in the top 4 menu positions, leading to locality set sizes of 1 to 4. On the other hand, search items in the *large locality set* scenario appear in all 8 menu

Locality Experiment		Non-Locality Experiment	
Small Locality Set	Large Locality Set	Mild Non-Locality	Strong Non-Locality
psroff -ms all	dbx search core	compress demos*	talk alee@db
refer paper.ms	man -k debugger	uncompress pics	rlogin front
spell abstract	search ins outs	rcp db:mbox tmp	ftp explorer
emacs paper.ms	ls -lt ins outs	grep talks mbox	mail -f mbox
wc conclusions	cd ~/tests/data	vi mbox oldmbox	ping spadina
lookbib biblio	mv a.out search	tar -f mail new	alias p more
sortbib biblio	cc -g search.c	chmod o-rwx tmp	which whoami
whereis psroff	lint ~/search.c	rm mail/db/jan*	rn -n can.ai
webster albeif	jove ~/search.c	file mail/db/a*	head dumplog
cp intro saved	cat -n search.c	zcat pics pics0	stty new crt
indxbib biblio	lpr -Pntx outs0	du -s mail/db/*	mkdir backup
p intro biblio	diff outs outs0	tail -f oldmbox	rlog letters

Table 1: The menu items used in the locality and non-locality experiment.

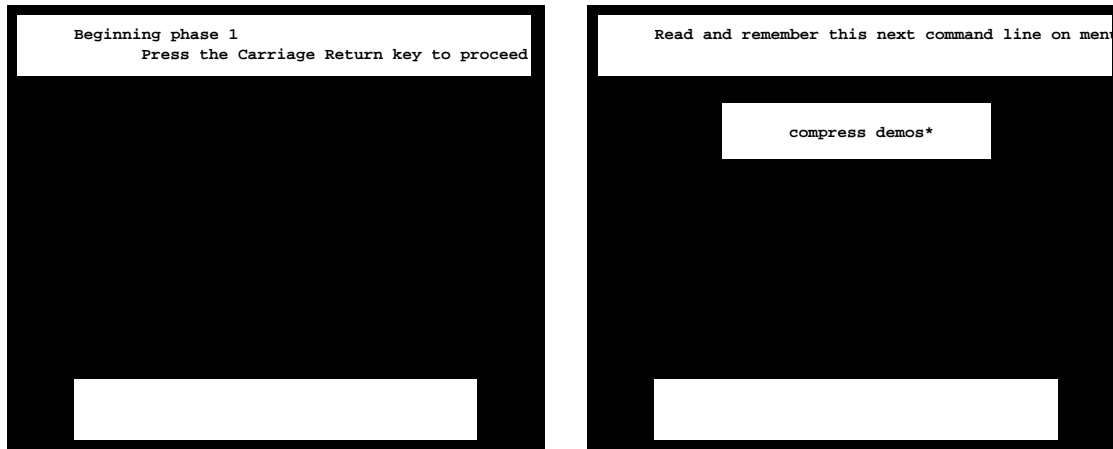


Figure 1: The two screen panels for the priming stage. The left screen panel marks the beginning of the priming stage. The right screen panel is representative of the screen panel when the list items are individually displayed for 3.5 seconds for the user to remember.

positions with diminishing probability, leading to locality set sizes 1 to 8. Table 2a lists the probability of finding a search item in each of the 8 serial positions for the two scenarios in this experiment.

In the non-locality experiment, two scenarios were examined: *mild non-locality* and *strong non-locality*. In the mild non-locality case, search items are absent 25% of the time and in the strong non-locality case, search items are absent 50% of the time. Serial position 0 represents an absent search item. The least recently referenced item is used as the search item in a trial requiring an absent search item. When the search item is present on the menu, the probability of finding it at serial position 2 to 8 is .28, .20, .16, .12, .10, .08, .06. However, since the percentage of the trials where the search item is present is different in the two scenarios, the effective probability of observing a search item is .75 and .50, respectively. Table 2b lists the effective probability of observing a search item in each of the serial positions of the menu. To ensure that no locality set biases occur in this experiment (i.e., no repeated reference to a set of the menu items), the probability of referencing a search item in position 1 is 0. As a result, there are 8 serial positions: positions 2 through 8 in the menu when the search item is present and position 0 when the search item is absent.

The probability distributions used in the scenario conditions were chosen to produce the desired locality and non-locality effects, subject to the constraint that there were sufficient test trials for each serial position.

Procedure

A session consists of a *practice block of test trials* and 4 blocks of test trials. Each block of trials begins with a *priming stage* (see left screen panel in Figure 1). Each of the 12 list items is presented individually for at least 3.5 seconds in the

Locality	Probability of Target at Serial Position							
	1	2	3	4	5	6	7	8
Small Locality Sets	.26	.24	.22	.20	.02	.02	.02	.02
Large Locality Sets	.18	.17	.15	.14	.12	.10	.08	.06

Table 2a: The probability of locating the target in serial positions 1 through 8 for the two scenarios in the locality experiment.

Non-Locality	Probability of Target at Serial Position							
	0	2	3	4	5	6	7	8
Mild Non-Locality	.25	.21	.15	.12	.09	.075	.06	.045
Strong Non-Locality	.50	.14	.10	.08	.06	.05	.04	.03

Table 2b: The probability of locating a search item in serial positions 0 and 2 to 8 for the two scenarios in the non-locality experiment. Serial position 0 represents the situation in which a search item does not appear on the menu.

reverse order of their initial position in the list, least recent first and most recent last (see right screen panel in Figure 1).

The priming stage is followed by a *searching stage* which consists of 100 timed trials (see top left and bottom right screen in Figure 2). On each trial, the search item is presented and is followed 2.5 seconds later by the presentation of the menu (see the top right and bottom left screen panels in Figure 2). If the search item appears in the menu, subjects are instructed to respond by hitting either the 'f' or 'j' key on the keyboard; whichever delimits the search item in the menu. This simple response scheme is used because it is crucial that the time to locate the item be accurately measured. In each trial, 'f' or 'j' is randomly assigned to each item of the menu, with equal probability. If the item is not located, subjects are instructed to respond by hitting the space bar on the keyboard. The time between the presentation of the menu and the subject's response is recorded and any errors that occur in the trial are noted.

Results and Discussion

The following sections describe the results and analyses of each of the two measured scores: number of errors and search times. These results are analyzed, for each experiment, using a two-factor, within-subjects ANOVA test.

Number of Errors

On average, less than 2.5 errors were committed in the 100 timed trials for each scenario condition, collapsed over all serial positions. The mean and standard error of this measure for each of the scenario conditions appear in Table 3.



Figure 2: The four panels associated with the searching stage. The top left panel and the bottom right panel mark the beginning and end of the searching stage. The top right panel is representative of the screen panel in which the search item to be searched for is presented for 2.5 seconds. The bottom left panel is representative of the screen panel in which the subject is to locate the search item in the displayed menu. If the subject locates the search item in the menu, the key for the letter delimiting the search menu item is pressed, otherwise, the space bar key is pressed.

There is no significant difference in the number of errors between the two levels of the scenario factor in either experiment (i.e., $F(1, 15) = 1.98$ $p = 0.18$ for the locality experiment and $F(1, 15) = 0.65$ $p = 0.43$ for the non-locality experiment).

In the locality experiment, the number of errors is significantly different for different serial positions, $F(7, 105) = 4.00$, $p < .001$. This result is erroneous and can be attributed to unequal sample sizes in the different conditions (i.e., fewer trials in serial positions 5 through 8 for the small locality set scenario compared to the large locality set scenario). A better estimate of search times in serial positions 5 through 8 is obtained for the small locality set scenario compared to the large locality set scenario.

Locality Mean \pm Std. Err. (N)		Non-Locality Mean \pm Std. Err. (N)	
Small Locality Sets	Large Locality Sets	Mild Non-Locality	Strong Non-Locality
2.4 \pm 0.4 (16)	1.7 \pm 0.4 (16)	2.4 \pm 0.4 (16)	1.9 \pm 0.5 (16)

Table 3: The mean and standard error of the number of errors observed for the 16 subjects in each of the scenario factors in the two experiments.

Search Times

For cognitive modelling purposes, it is sufficient to examine the average visual search performance profile (i.e., to examine average subject performance times). In the same spirit as other cognitive modelling research based on MHP, an approximation of the search strategy and the quantitative relationship between search times and serial position are determined. Therefore, the focus is on the mean search times. This is based on the mean search times of each subject at each serial position in each of the scenario conditions.

Figures 3 and 4 plot, for the locality and non-locality experiment, the mean search times as a function of the 8 serial positions for each of the two scenario conditions in each experiment. As suggested by the respective plots, the main effect of serial position on search time is highly significant in both the locality ($F(7,105) = 60.6$, $p < .0001$) and the non-locality experiments ($F(7,105) = 47.9$, $p < .0001$). This suggests that a relationship exists between search time and serial position of search items. In order to determine the exact nature of this relationship, a one-factor within-subjects trend analysis was performed on each scenario condition in each experiment.

In the locality experiment case, there is a significant linear relationship in the small locality set scenario ($F(1,105) = 134.9$, $p < .0001$) and in the large locality set scenario ($F(1,105) = 448.2$, $p < .0001$). However, there is also a significant quadratic relationship in the large locality set scenario ($F(1,105) = 14.0$, $p < .01$). A closer examination of the plot suggests that this may be caused by the low search time for serial position 1. This low search time for serial position 1 suggests that a subject's skill in recognizing the same search items which appear in back-to-back trials is more advanced than their skill in recognizing the same search items which are separated by more than one trial. Thus, subjects may just proceed to make an immediate response without the need to examine the menu items. To confirm this conjecture, a trend analysis was performed again on the data but ignoring the search times for serial position 1. The test reveals a significant linear relationship for each scenario condition (both in the small locality set scenario, $F(1,90) = 66.7$, $p < .0001$ and in the large locality set scenario, $F(1,90) = 216.8$, $p < .0001$) and the lack of a significant quadratic relationship in either scenario. We therefore, treat serial position 1 in both scenarios as special cases. This special treatment of serial position 1 is applied to both scenarios in order to ensure equal treatment.

A trend analysis was also performed on each scenario of the non-locality experiment for search times in serial positions 2 through 8. Serial position 0 is

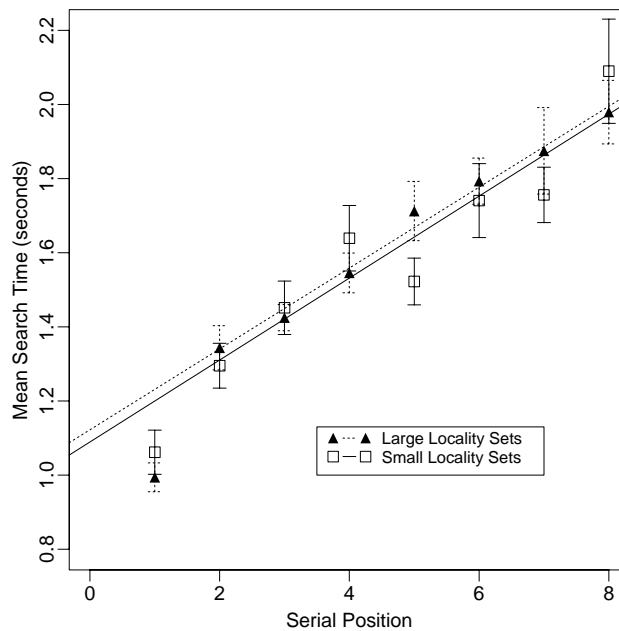


Figure 3: Locality: A plot of the mean search times (based on the mean search times of the individual subjects) as a function of serial position along with the fitted line. Note: the data for serial position 1 was not included in the data used to derive the fitted line because a strong practice effect in this serial position produced a depressed reaction time.

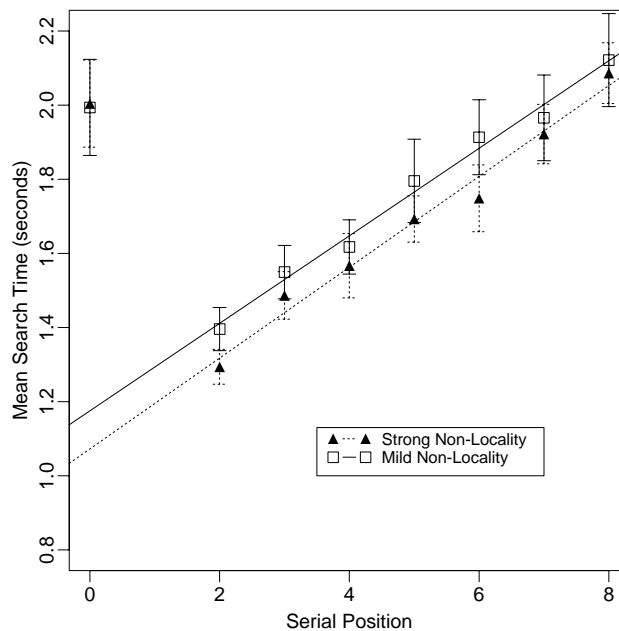


Figure 4: Non-Locality: A plot of the mean search times (based on the mean search times of the individual subjects) as a function of serial position along with the fitted line.

treated as a special case and is not included in the trend analysis because a subject's response in this case is different from the case when the search item appears on the menu. In fact, the response scheme in the latter case is simpler and requires less time than the former case. This is confirmed in an examination of search times for this serial position when compared to search times at the other serial positions. For both scenarios, there is only a significant linear relationship between search times and serial position (in the mild non-locality scenario, $F(1,90) = 200.1$, $p < .0001$ and in the strong non-locality scenario, $F(1,90) = 187.3$, $p < .0001$).

Table 4 lists the mean search times for the special cases in each of the two experiments. Based on the results of the trend analysis, we performed a least squares linear regression on the search times against serial positions 2 through 8 for each scenario condition in the two experiment (see Figures 3 and 4). From the regression, the slope and the intercept for each fitted line as well as the coefficient of determination¹, r^2 , is determined (see Table 5).

There are four fitted lines, one for each scenario condition.

$$\text{Task Performance Time} = \text{Search Rate} * I + \text{Aggregate Time} \quad \text{Eqn. (1)}$$

The slope of the fitted line is the incremental time needed to search each additional menu item (i.e., Search Rate). The intercept is the Aggregate Times of the processes that occur in every test trial (e.g., start and stop times of visual scan, decision time, response time). As shown in Table 5, search rates remain relatively unchanged across the two scenario conditions in each experiment. Furthermore, search rates between the two experiments are nearly identical (.11 secs./item in the locality experiment and .12 secs./item in the non-locality experiment). The main differences are attributed to intercept values. This means that each scenario condition involves cognitive operations of different durations (see Table 5).

The results of this experiment indicate that the time to search a history menu is approximated by a linear function that varies directly with the number of items to search (i.e., the menu search is serial). The search rate for both the locality and non-locality experiments are nearly identical with aggregate times that are slightly different for the different scenario conditions. The history lists in our experiment are typical history lists. Therefore, the results are generalizable to user histories whose references exhibit locality and non-locality effects.

Time Parameter for the Pure Menu Search Operator

Eqn. (1) includes a menu search component and a response component. This response component appears in the intercept – Aggregate Time – because the response task is one of the subtasks that is performed in every test trial. Because the analyses in Chapter 6 consider task proposals which use different response methods, the response time in the menu search task is separated out from the task performance time to yield a pure menu search time.

¹ r^2 represents the proportion of the sum of squares of deviations of the y-values about their mean that can be attributed to a linear relationship between y and x.

Experiment	Scenario	Serial Position	Search Times (secs.) Mean \pm Std. Err.
Locality	Small Locality Set	1	1.06 \pm .06
	Large Locality Set	1	.99 \pm .04
Non-Locality	Mild Non-Locality	0	1.99 \pm .13
	Strong Non-Locality	0	2.00 \pm .12

Table 4: The mean search times, along with the standard error, for the special cases of the locality and non-locality experiment.

Experiment	Scenario	Search Rates (secs/position) Slope \pm Std. Err.	Aggregate Times (seconds) Intercept \pm Std. Err.	r^2
Locality	Small Locality Set	.11 \pm .02	1.09 \pm .10	.87
	Large Locality Set	.11 \pm .01	1.12 \pm .03	.99
Non-Locality	Mild Non-Locality	.12 \pm .01	1.18 \pm .03	.99
	Strong Non-Locality	.12 \pm .01	1.07 \pm .04	.98

Table 5: The slope and intercept of a weighted regression fit on the data shown in Figures 3 and 4 for each scenario factor. The slope is the search rate and the intercept is the aggregate duration of the processes that occur regardless of serial position.

Figure 5 contains a ARTLESS description of the menu search task. This description yields another form of Eqn. (1) for the time to perform this task:

$$\text{Task Performance Time} = P_{S_0} + IP_{S_1} + p_f(P_I + 3.5C_E + M_B) + (1 - p_f)(3C_E + M_B)$$

where p_f is the probability of the search item being on the menu while $(1 - p_f)$ is the probability of the search item not being on the menu. In the two scenario conditions of the locality experiment p_f is 1. In the non-locality experiment, p_f is .75 for the mild non-locality condition and p_f is .5 for the strong non-locality condition. These p_f values are not in one-to-one correspondence to those observed in Chapter 5. The reason is that, unlike a real user session, a laboratory experiment is a small scale simulation of a user session. As a result, we had to amplify the values for p_f in order to reproduce, in a laboratory experiment, the locality and non-locality effects observed in our behavioural studies (i.e., small and large locality set effects and mild and strong non-locality effects).

$P_S(l)$ is the time associated with the non-response activities. P_{S_0} is the Aggregate Time – Response Time and P_{S_1} is the Search Rate.

$$P_S(l) = P_{S_0} + IP_{S_1}$$

```

L1   $P_S(l)$  Match_Result  $\leftarrow$  Search (Target)
L2   $C_E$     IF-SUCCEEDED Found? (Match_Result) THEN BEGIN
L3   $P_I$     Letter  $\leftarrow$  Perceive-Letter (<LETTER>)
L4   $C_E$     IF-SUCCEEDED Is_F? (Letter) THEN BEGIN
L5   $C_E$     Initiate (<LETTER>)
L6   $M_B$     Key (<LETTER>)
L7  END
L8   $C_E$     ELSE IF-SUCCEEDED Is_J? (Letter) THEN BEGIN
L9   $C_E$     Initiate (<LETTER>)
L10  $M_B$     Key (<LETTER>)
L12 END
L13 END
L14  $C_E$     ELSE IF-SUCCEEDED Not_Found? (Item) THEN BEGIN
L15  $C_E$     Initiate (<SPACE>)
L16  $M_B$     Key (<SPACE>)
L17 END

```

$$\begin{aligned}
\text{Total Cost} &= P_S(l) + p_f(C_E + P_I + .5(2C_E + M_B) + .5(3C_E + M_B)) \\
&\quad + (1 - p_f)(3C_E + M_B) \\
&= P_S(l) + p_f(P_I + 3.5C_E + M_B) + (1 - p_f)(3C_E + M_B) \\
&= P_{S_0} + lP_{S_1} + p_f(P_I + 3.5C_E + M_B) + (1 - p_f)(3C_E + M_B)
\end{aligned}$$

Figure 5: A description of the menu search task and the associated performance time. The response component of the task is described by lines L2 to L17. The ‘f’ and ‘j’ letters appear with equal probabilities (i.e., .5). A response involving either the ‘f’ or ‘j’ key takes one of two routes: a) positive test for presence of ‘f’ character first (L4 to L7) or b) negative test for presence of ‘f’ and then a positive test for presence of ‘j’ (L4, L8 to L12). p_f is the probability of the search item being on the menu while $(1 - p_f)$ is the probability of the search item not being on the menu. There are two routes for determining the correct response: a) positive test for presence of menu item (L2 to L13) or b) negative test for presence of menu item and then positive test for absence of menu item (L2, L14 to L17). p_f is 1 for both scenario conditions of the locality experiment. In the non-locality experiment, p_f is .75 for the mild non-locality condition and .5 for the strong non-locality condition.

The response time is made up of two component response times; one for each of the two selection schemes. The first selection scheme is used when the search item is on the menu and the subject must press either the ‘f’ or ‘j’ key, whichever appears next to the search item. The performance time is $P_I + 3.5C_E + M_B$. The second selection scheme is used when a search item is not on the menu and the subject must press the ‘space bar’. This performance time is $3C_E + M_B$.

All subjects are familiar with the keyboard so M_B is equivalent to the motor processor cycle time – 70 msec. Similarly, one perceptual processor cycle, $P_I = 100$ msec., is needed to notice the letter delimiting the target. Finally, each mental execution step involves one basic cognitive cycle $C_E = 70$ msec.

The resulting response time is 420 msec when the search item appears on the menu and 280 msec when the search item does not appear on the menu. Table 6 lists the aggregate time, the response time, and the pure menu search time for each scenario condition in each of the experiments.

Concluding Remarks

Our experiment to estimate the menu search operator considered visual search performance with a menu organized in a least-recently used (LRU for short) order; the top-most menu item is the most recently referenced command while the bottom-most menu item is less recently referenced. Menu references in the experiment exhibited either locality or non-locality behaviours. Results reveal that menu search times are approximated by a linear function of the serial position of the search item in the menu.

Because we did not run subjects for all combinations of the two scenario conditions in the locality case with the two scenario conditions in the non-locality case, we cannot make a statistical comparison between the results of the locality and non-locality cases. However, an informal comparison of the menu search rates for the locality and non-locality case reveals nearly identical values. Any differences in menu search times within the two scenario conditions of the locality and non-locality cases are attributed to nuances in the particular scenarios resulting in cognitive operators of different durations [Chase, 1986].

Experiment	Scenario	Aggregate Time (msecs.)	Response Time (msecs.)	Pure Search Time (msecs.)
Locality	Small Locality Set	1090	420	670 ± 110 /
	Large Locality Set	1120	420	700 ± 110 /
	Mean	1110	420	690 ± 110 /
Non-Locality	Mild Non-Locality	1180	380	800 ± 120 /
	Strong Non-Locality	1070	320	750 ± 120 /
	Mean	1130	350	780 ± 120 /

Table 6: The Aggregate Time from Table 5, the response time, and the pure search time associated with each scenario condition of each experiment and the average case.

Appendix E

Task Descriptions for Proposed Designs

Command Feature

!! – Repeat the Most Recent Command

C_M Recency("Command Line")

C_M Get-!! (<!!>)

C_E Initiate (<!>)

M_K Key (<!>)

C_E Initiate (<!>)

M_K Key (<!>)

C_E Initiate (<RETURN>)

M_K Key (<RETURN>)

$$C_M + C_M + C_E + M_K + C_E + M_K + C_E + M_K$$

$$2C_M + 3C_E + 3M_K$$

Assumptions:

- (1) Recency("Command Line") performed in a generic memory retrieval operator cycle time.

!name – Repeat Most Recent Command with Given Name

```

 $C_M$   Get-! (<!>)
 $C_E$   Initiate (<!>)
 $M_K$   Key (<!>)
 $C_E$   Command-Name  $\leftarrow$  Get-Name("Command Name")
      Count  $\leftarrow$  1
      REPEAT BEGIN
 $C_E$       Initiate (Command-Name[Count])
 $M_K$       Key (Command-Name[Count])
      Count  $\leftarrow$  Count + 1
      UNTIL Is-Equal?(Count,  $\bar{n}_0$ )
 $C_E$   IF-SUCCEEDED Is-Equal? (Count,  $\bar{n}_0$ ) THEN BEGIN
 $C_E$       Initiate (<RETURN>)
 $M_K$       Key (<RETURN>)
      END

 $C_M + C_E + M_K + C_E + \bar{n}_0(C_E + M_K) + C_E + C_E + M_K$ 

 $C_M + (\bar{n}_0 + 4)C_E + (\bar{n}_0 + 2)M_K$ 

```

Assumptions:

- (1) No memory retrieval required to Get-Name("Command Name") because the *command* is in a user's working memory, just a mental step to obtain the *command name*.
- (2) \bar{n}_0 is the average length of the *command name*.

!number – Repeat Command with Given Number

C_M	Number \leftarrow Get-Number("Command Line")
C_M	Get-!n (<!n>)
C_E	Initiate (<!>)
M_K	Key (<!>)
C_E	Initiate (Number[First-Digit])
M_K	Key (Number[First-Digit])
C_E	Initiate (Number[Second-Digit])
M_K	Key (Number[Second-Digit])
C_E	Initiate (Number[Third-Digit])
M_K	Key (Number[Third-Digit])
C_E	Initiate (<RETURN>)
M_K	Key (<RETURN>)

$$C_M + C_M + C_E + M_K + 3(C_E + M_K) + C_E + M_K$$

$$2C_M + 5C_E + 5M_K$$

Assumptions:

- (1) Users retain an association between the *command* and its number. Therefore, to recall the *number*, users retrieve the association.
- (2) The most optimal method was selected, associating *number* with *command*, rather than other methods which require significantly more steps. For example, users can find the number by requesting a listing of their history list, scanning the list to find the *command*, and then noting the associated *number*.
- (3) A typical session in which lots of work is being done would involve 100 to 999 commands. Thus, on the average, there are 3 digits to a *command*.

!?*argument* – Repeat Most Recent Command with Given Argument

```

CM  Get-!<?> (<!>)
CE  Initiate (<!>)
MK  Key (<!>)
CE  Initiate (<?>)
MK  Key (<?>)
CM  Get-One-Argument ("Argument")
      Count ← 1
      REPEAT BEGIN
CE      Initiate (Argument[Count])
MK      Key (Argument[Count])
          Count ← Count + 1
      UNTIL Is-Equal?(Count,  $\bar{n}_A$ )
CE  IF-SUCCEEDED Is-Equal? (Count,  $\bar{n}_A$ ) THEN BEGIN
CE      Initiate (<RETURN>)
MK      Key (<RETURN>)
      END

```

$$C_M + C_E + M_K + C_E + M_K + C_M + \bar{n}_A(C_E + M_K) + C_E + C_E + M_K$$

$$2C_M + (\bar{n}_A + 4)C_E + (\bar{n}_A + 3)M_K$$

Assumptions:

- (1) A mental retrieval is required to determine a suitable argument in the *command* which will retrieve the most recent copy of the desired *command*.
- (2) The whole *argument* must be provided and not a substring of it.
- (3) \bar{n}_A is the average length of a command's *argument*.

History Menu

Popup – Menu Appear on User Request

C_M	Get-PopUpMenu (<MOUSE>)
C_E	Initiate (<MOUSE>)
M_H	Home (<MOUSE>)
C_E	Initiate (<DRAG>)
$M_{D_M}(l)$	Drag ("Item at Position l ")
S	Wait-System-Response(<MENU>)
$P_S(l)$	Match ("Item at Position l ")
C_E	Initiate (<KEYBOARD>)
M_H	Home (<KEYBOARD>)

$$C_M + C_E + M_H + C_E + \sum_{l=1}^8 p_l M_{D_M}(l) + S + \sum_{l=1}^8 p_l P_S(l) + C_E + M_H$$

$$C_M + 2M_H + 3C_E + S + \sum_{l=1}^8 p_l (P_S(l) + M_{D_M}(l))$$

Assumptions:

- (1) In estimates of Drag("Item at Position l "), the time takes into account the button down to cause menu to appear, drag through to the desired item, and then release of button to make selection.

Static – Menu Remains on Screen

C_M	Get-StaticMenu (<MOUSE>)
P_E	Eye-Move (<MENU>)
$P_S(I)$	Match ("Item at position I ")
C_E	Initiate (<MOUSE>)
M_H	Home (<MOUSE>)
C_E	Initiate (<POINT>)
$M_{P_M}(I)$	Point ("Item at Position I ")
C_E	Initiate (<KEYBOARD>)
M_H	Home (<KEYBOARD>)

$$C_M + P_E + \sum_{I=1}^8 p_I P_S(I) + C_E + M_H + C_E + \sum_{I=1}^8 p_I M_{P_M}(I) + C_E + M_H$$

$$C_M + 2M_H + 3C_E + P_E + \sum_{I=1}^8 p_I (P_S(I) + M_{P_M}(I))$$

Assumptions:

- (1) In estimates of Point("Item at Position I "), the movement time takes into account the time to get to the desired item, and a button down and up for the selection.
- (2) To find the static menu, users must first move their eyes to region where the static menu is. P_E is 230 msec which represents a typical eye movement value.
- (3) No system time because menu remains on the screen.

Step Buffer

Single Step – Visually Examine Each Item Individually

```

CM   Get-StepMenu (<STEP MENU>)
PS0 Prep-Search (<STEP MENU>)
      REPEAT BEGIN
CE     Initiate (<STEP MENU>)
MK     Key (<STEP MENU>)
S       Wait-System-Response(<STEP MENU>)
PI     Command-Line ← Perceive-Text ("Command Line")
PS1    Match-Result ← Match (Command-Line)
      UNTIL Found? (Match-Result)
CE     IF-SUCCEEDED Found? (Match-Result) THEN BEGIN
CE       Initiate (<RETURN>)
MK       Key (<RETURN>)
      END

```

$$C_M + P_{S_0} + \sum_{l=1}^8 lp_l(C_E + M_K + S + P_I + P_{S_1}) + C_E + C_E + M_K$$

$$C_M + P_{S_0} + \sum_{l=1}^8 lp_l(S + P_I + P_{S_1}) + \left(\sum_{l=1}^8 lp_l + 2\right)C_E + \left(\sum_{l=1}^8 lp_l + 1\right)M_K$$

Assumptions:

- (1) <STEP MENU> key is a cursor key available on the keyboard. The time to press this key is equivalent to the time to press any other key on the keyboard, M_K .

Recall & Step – Recall How Recent and Step Right to It

```

CM  Recency("Command Line")
CM  Get-StepMenu (<STEP MENU>)
PS0  Prep-Search (<STEP MENU>)
      Count ← 1
      REPEAT BEGIN
CE          Initiate (<STEP MENU>)
MK          Key (<STEP MENU>)
              Count ← Count + 1
      UNTIL Is-Equal? (Count, 1)
CE  IF-SUCCEEDED Is-Equal? (Count, 1) THEN BEGIN
S          Wait-System-Response(<STEP MENU>)
PI        Command-Line ← Perceive-Text ("Command Line")
PS1      Match-Result ← Match (Command-Line)
CE        IF-SUCCEEDED Found? (Match-Result) THEN BEGIN
CE          Initiate (<RETURN>)
MK          Key (<RETURN>)
      END
END

```

$$C_M + C_M + P_{S_0} + \sum_{l=1}^8 l p_l (C_E + M_K) + C_E + S + P_I + P_{S_1} + C_E + C_E + M_K$$

$$2C_M + P_{S_0} + P_{S_1} + S + P_I + \left(\sum_{l=1}^8 l p_l + 3\right) C_E + \left(\sum_{l=1}^8 l p_l + 1\right) M_K$$

Assumptions:

- (1) Recency("Command Line") performed in a generic memory retrieval operator cycle time.
- (2) <STEP MENU> key is a cursor key available on the keyboard. The time to press this key is equivalent to the time to press any other key on the keyboard, M_K .

History Completion on Command Name

```

 $C_M$   Get-StepMenu (<Esc>)
 $C_E$   Command-Name  $\leftarrow$  Get-Name("Command Name")
      Count  $\leftarrow$  1
      REPEAT BEGIN
 $C_E$       Initiate (Command-Name[Count])
 $M_K$       Key (Command-Name[Count])
      Count  $\leftarrow$  Count + 1
      UNTIL Is-Equal?(Count,  $\bar{n}_0$ )
 $C_E$   IF-SUCCEEDED Is-Equal? (Count,  $\bar{n}_0$ ) THEN BEGIN
      REPEAT BEGIN
 $C_E$       Initiate (<Esc>)
 $M_K$       Key (<Esc>)
 $S$       Wait-System-Response(<Esc>)
 $P_I$       Command-Line  $\leftarrow$  Perceive-Text ("Command Line")
 $P_{S_1}$      Match-Result  $\leftarrow$  Match (Command-Line)
      UNTIL Found? (Match-Result)
 $C_E$       IF-SUCCEEDED Found? (Match-Result) THEN BEGIN
 $C_E$       Initiate (<RETURN>)
 $M_K$       Key (<RETURN>)
      END
      END

```

$$C_E + \bar{n}_0(C_E + M_K) + C_E + C_M + e(C_E + M_K + S + P_I + P_{S_1}) + C_E + C_E + M_K$$

$$(\bar{n}_0 + 4)C_E + (\bar{n}_0 + 1)M_K + C_M + e(C_E + M_K + S + P_I + P_{S_1})$$

$$C_M + e(S + P_I + P_{S_1}) + (\bar{n}_0 + e + 4)C_E + (\bar{n}_0 + e + 1)M_K$$

Assumptions:

- (1) <Esc> key is a cursor key available on the keyboard. The time to press this key is equivalent to the time to press any other key on the keyboard, M_K .
- (2) e items are assumed to be examined.

Typing

```

CE   Command-Name ← Get-Name("Command Name")
Count ← 1
REPEAT BEGIN
CE       Initiate (Command-Name[Count])
MK       Key (Command-Name[Count])
Count ← Count + 1
UNTIL Is-Equal?(Count,  $\bar{n}_0$ )
CE   IF-SUCCEEDED Is-Equal? (Count,  $\bar{n}_0$ ) THEN BEGIN
CE       Argument ← Get-Next-Word ("First Argument")
CE       IF-SUCCEEDED An-Argument? (Argument) THEN BEGIN
REPEAT BEGIN
CE           Initiate (<SPACE>)
MK           Key (<SPACE>)
Count ← 1
REPEAT BEGIN
CE               Initiate (Argument[Count])
MK               Key (Argument[Count])
Count ← Count + 1
UNTIL Is-Equal?(Count,  $\bar{n}_A$ )
CE           IF-SUCCEEDED Is-Equal? (Count,  $\bar{n}_A$ ) THEN BEGIN
A ← A + 1
END
CE           Argument ← Get-Next-Word ("Next Argument")
UNTIL No-Argument? (Argument)
CE           IF-SUCCEEDED No-Argument? (Argument) THEN BEGIN
CE               Initiate (<RETURN>)
MK               Key (<RETURN>)
END
END
END
CE   ELSE IF-SUCCEEDED No-Argument? (Argument) THEN BEGIN
CE       Initiate (<RETURN>)
MK       Key (<RETURN>)
END
END

```

$$C_E + \bar{n}_0(C_E + M_K) + C_E + C_E + C_E + p_a(C_E + C_E + M_K) + (1 - p_a)(a(C_E + M_K + \bar{n}_A(C_E + M_K) + C_E + C_E) + C_E + C_E + M_K)$$

$$4C_E + \bar{n}_0(C_E + M_K) + 2C_E + M_K + a(1 - p_a)(3C_E + M_K + \bar{n}_A(C_E + M_K))$$

$$(6 + \bar{n}_0)C_E + (1 + \bar{n}_0)M_K + a(1 - p_a)((3 + \bar{n}_A)C_E + (1 + \bar{n}_A)M_K)$$

Assumptions:

- (1) No mental retrieval operator is required in this task as the *command* is in a user's working memory.
- (2) \bar{n}_0 is the average command *name* length.
- (3) \bar{n}_A is the average command *argument* length.

Appendix F

Task Descriptions for Experimental Tasks

Recall By Argument

P_I Task \leftarrow Perceive-Task ("Task Request")
 C_{CRT} ChoiceReactionTime(Task)
 C_M Get-!<?>
 C_E Initiate (<!>)
 M_K Key (<!>)
 C_E Initiate (<?>)
 M_K Key (<?>)
 C_M Argument \leftarrow Get-One-Argument("Argument")
 Count \leftarrow 1
 REPEAT BEGIN
 C_E Initiate (Argument[Count])
 M_K Key (Argument[Count])
 Count \leftarrow Count + 1
 UNTIL Is-Equal?(Count, n_A)
 C_E IF-SUCCEEDED Is-Equal? (Count, n_A) THEN BEGIN
 C_E Initiate (<RETURN>)
 M_K Key (<RETURN>)
 END

$$P_I + C_{CRT} + C_M + C_E + M_K + C_E + M_K + C_M + n_A(C_E + M_K) + 2C_E + M_K$$

$$P_I + C_{CRT} + 2C_M + (n_A + 4)C_E + (n_A + 3)M_K$$

Assumptions:

- (1) A mental retrieval is required to determine a suitable argument in the *command* which will retrieve the most recent copy of the desired *command*.
- (2) The whole *argument* must be provided and not a substring of it.
- (3) n_A is the length of a command's *argument*.

Popup Menu

P_I	Task ← Perceive-Task ("Task Request")
C_{CRT}	ChoiceReactionTime(Task)
C_M	Get-PopUpMenu (<MOUSE>)
C_E	Initiate (<MOUSE>)
M_H	Home (<MOUSE>)
C_E	Initiate (<DRAG>)
$M_{D_M}(I)$	Drag ("Matching Item")
S	Wait-System-Response(<MENU>)
$P_S(I)$	Match ("Item at Position I ")
	$P_I + C_{CRT} + C_M + C_E + M_H + C_E + M_{D_M}(I) + S + P_S(I)$
	$P_I + C_{CRT} + C_M + 2C_E + M_H + M_{D_M}(I) + S + P_S(I)$

Assumptions:

- (1) In estimates of Drag("Item at Position I "), the time takes into account the button down to cause menu to appear, drag through to the desired item, and then release of button to make selection.

Single Step Buffer

```

PI    Task ← Perceive-Task ("Task Request")
CCRT  ChoiceReactionTime(Task)
CM    Get-StepMenu (<STEP MENU>)
PS0  Prep-Search (<STEP MENU>)
      REPEAT BEGIN
CE      Initiate (<STEP MENU>)
MB      Key (<STEP MENU>)
PI      Command-Line ← Perceive-Text ("Command Line")
PS1    Match-Result ← Match (Command-Line)
      UNTIL Found? (Match-Result)
CE    IF-SUCCEEDED Found? (Match-Result) THEN BEGIN
CE      Initiate (<RETURN>)
MK      Key (<RETURN>)
      END

PI + CCRT + CM + PS0 + I(CE + MB + PI + PS1) + CE + CE + MK

PI + CCRT + CM + PS0 + IPS1 + IPI + (I + 2)CE + IMB + MK

PI + CCRT + CM + PS(I) + IPI + (I + 2)CE + IMB + MK

```

Assumptions:

- (1) In the experiment, the subject's hands were already positioned over the keys. The keyboard interaction is limited primarily to pressing the <STEP MENU> key. Thus, the time to press the <STEP MENU> key is the time to press a button, M_B .
- (2) However, the time to press the <RETURN> key is equivalent to the time to perform a keystroke operator, M_K

Typing

```

PI   Task ← Perceive-Task ("Task Request")
CCRT ChoiceReactionTime(Task)
CE   Command-Name ← Get-Name("Command Name")
      Count ← 1
      REPEAT BEGIN
CE       Initiate (Command-Name[Count])
MK       Key (Command-Name[Count])
      Count ← Count + 1
      UNTIL Is-Equal?(Count, n0)
CE   IF-SUCCEEDED Is-Equal? (Count, m) THEN BEGIN
      A ← 1
      REPEAT BEGIN
CE       Initiate (<SPACE>)
MK       Key (<SPACE>)
CE       Argument ← Get-Argument ("Next Argument")
      Count ← 1
      REPEAT BEGIN
CE       Initiate (Argument[Count])
MK       Key (Argument[Count])
      Count ← Count + 1
      UNTIL Is-Equal?(Count, nA)
CE   IF-SUCCEEDED Is-Equal? (Count, nA) THEN BEGIN
      A ← A + 1
      END
      UNTIL Is-Greater? (A, a)
CE   IF-SUCCEEDED Is-Greater? (A, a) THEN BEGIN
CE       Initiate (<RETURN>)
MK       Key (<RETURN>)
      END
      END
      END

```

$$P_I + C_{CRT} + C_E + n_0(C_E + M_K) + C_E + \left(\sum_{A=1}^a (C_E + M_K + C_E + n_A(C_E + M_K) + C_E) + C_E + C_E + M_K \right)$$

$$P_I + C_{CRT} + (1 + n_0 + 1 + a + a + \sum_{A=1}^a n_A + a + 2)C_E + (n_0 + a + \sum_{A=1}^a n_A + 1)M_K$$

$$P_I + C_{CRT} + (4 + 3a + \sum_{A=0}^a n_A)C_E + (1 + a + \sum_{A=0}^a n_A)M_K$$

Assumptions:

- (1) n_0 is the length of a command *name*.
- (2) n_A is the length of a command *argument*.

Bibliography

- Agre, P.E. & Chapman, D. (1990). What are plans for?. In P. Maes (Ed.), *New Architectures for Autonomous Agents: Task-Level Decomposition and Emergent Functionality*. Cambridge, Ma.: MIT Press.
- Akin, O., Baykan, C., & Rao, D.R. (1987, March). Structure of a directory space: A case study with a UNIX operating system. *International Journal of Man-Machine Studies*, 26(3), 361–382.
- Anderson, J.R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89(4), 369–406.
- Archer, J.E., Jr., Conway, R., & Schneider, F.B. (1984, January). User recovery and reversal in interactive systems. *ACM Transactions on Programming Languages and Systems*, 6(1), 1–19.
- Ash, W.L. (1981a, August). Mxec: Parallel processing with an advanced macro facility. *Communications of the ACM*, 24(8), 502–509.
- Bannon, L., Cypher, A., Greenspan, S., & Monty, M.L. (1983, December 12–15). Evaluation and analysis of users' activity organization. In A. Janda (Ed.), *Proceedings of the CHI'83 Human Factors in Computer Systems* (pp. 54–57). New York: Association for Computing Machinery.
- Barnes, D.J. & Bovey, J.D. (1986, September). Managing command submission in a multiple-window environment. *Software Engineering Journal*, 1(5), 177–184.
- Benyon, D. (1984, September 4–7). MONITOR. A self-adaptive user interface. In B. Shackel (Ed.), *Interact'84: Proceedings of the First IFIP Conference on Human-Computer Interaction* (Vol. 1, pp. 207–313). Amsterdam: North-Holland.
- Bier, E. & Freedman, M. (1985). *PlayerPiano: Playback scripts for window systems* (unpublished paper).
- Bobker, S. (1988, January). Command performance. *MacUser*, 4(1), 114–122.
- Boies, S.J. (1974). User behavior on an interactive computer system. *IBM Systems Journal*, 13(1), 2–18.
- Bourne, S.R. (1979, January 10). *An Introduction to the UNIX shell* (UNIX Programmer's Manual (7th ed.) Volume 2: Supplementary Documentation (Getting Started)).

- Brown, J.S. & Newman, S.E. (1985). Issues in cognitive and social ergonomics: From our house to Bauhaus. *Human-Computer Interaction*, 1(4), 359–391.
- Card, S.K. & Henderson, D.A., Jr. (1987, April 5–9). A multiple, virtual-workspace interface to support user task switching. In J.M. Carroll & P.P. Tanner (Eds.), *Proceedings of the CHI+GI'87 Human Factors in Computing Systems and Graphics Interface* (pp. 53–59). New York: Association for Computing Machinery.
- Card, S.K. (1984). Visual search of computer command menus. In H. Bourma & D.G. Bouwhuis (Eds.), *Attention and Performance X: Control of Language Processes* (pp. 97–108). London: Lawrence Erlbaum Associates.
- Card, S.K. (1989). Human Factors and Artificial Intelligence. In P.A. Hancock & M.H. Chignell (Eds.), *Intelligent Interfaces: Theory, Research and Design* (pp. 27–48). New York: North-Holland.
- Card, S.K., Moran, T.P., & Newell, A. (1980, July). The keystroke level model for user performance time with interactive systems. *Communications of the ACM*, 23(7), 396–410.
- Card, S. K., Moran, T.P., & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ.:Lawrence Erlbaum Associates.
- Card, S.K., Moran, T.P., & Newell, A. (1986). The model human processor: An engineering model of human performance. In K.R. Boff, L. Kaufman, J.P. Thomas (Ed.), *Handbook of Perception and Human Performance: Vol 2. Cognitive Processes and Performance* (pp. 45.1–45.35). New York: Wiley.
- Card, S.K., Pavel, M., & Farrell, J.E. (1984, September 4–7). Window-based computer dialogues. In B. Shackel (Ed.), *Interact'84: Proceedings of the First IFIP Conference on Human-Computer Interaction* (Vol. 1, pp. 355–359). Amsterdam: North-Holland.
- Carey, T. (1982, November). User differences in interface design. *Computer*, 15(11), 14–20.
- Carroll, J.M., Kellogg, W.A., & Rosson, M.B. (1991). The task-artifact cycle. In J.M. Carroll (Ed.), *Designing Interaction: Psychology at the Human-Computer Interface* (pp. 74–102). Cambridge: Cambridge University Press.
- Chan, P.P. (1984, July). *Learning Considerations in User Interface Design: The Room Model* (Technical Report CS-84-16). Waterloo, Ontario, Canada: University of Waterloo, Computer Science Department.
- Chase, W.G. (1986). Visual information processing. In K.R. Boff, L. Kaufman, J.P. Thomas (Ed.), *Handbook of Perception and Human Performance: Vol 2. Cognitive Processes and Performance* (pp. 28.1–28.71). New York: Wiley.
- Chin, D.N. (1986, April 13–17). User modeling in UC, the UNIX Consultant. In M. Mantei & P. Orbeton (Eds.), *Proceedings of the CHI'86 Human Factors in Computer Systems* (pp. 24–28). New York: Association for Computing Machinery.
- Cowan, W.B. & Wein, M. (1990, August 27–31). State versus history in user interfaces. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Interact'90. Proceedings of the Third IFIP Conference on Human-Computer Interaction*

- (pp. 555–560). Amsterdam: North-Holland.
- Croft, W.B. (1984, February). The role of context and adaptation in user interfaces. *International Journal of Man-Machine Studies*, 21(4), 283–292.
- Cypher, A. (1986). The structure of users' activities. In D.A. Norman & S.W. Draper (Eds.), *User-Centered System Design: New Perspectives on Human-Computer Interaction* (pp. 243–263). Hillsdale, NJ.: Lawrence Erlbaum Associates.
- Denning, P.J. (1980, January). Working sets past and present. *IEEE Transactions on Software Engineering*, SE-6(1), 64–84.
- Desmarais, M.C. & Pavel, M. (1987, September 1–4). User knowledge evaluation: An experiment with UNIX. In H.J. Bullinger, B. Shackel, & K. Kornwachs (Eds.), *Interact'87. Proceedings of the Second IFIP Conference on Human-Computer Interaction* (pp. 151–156). Amsterdam: North-Holland.
- Draper, S.W. (1984, September 4–7). The nature of expertise in UNIX. In B. Shackel (Ed.), *Interact'84: Proceedings of the First IFIP Conference on Human-Computer Interaction* (Vol. 2, pp. 182–186). Amsterdam: North-Holland.
- Draper, S.W. (1986). Display managers as the basis for user-machine communication. In D.A. Norman & S.W. Draper (Eds.), *User-Centered System Design: New Perspectives on Human-Computer Interaction* (pp. 339–352). Hillsdale, NJ.: Lawrence Erlbaum Associates.
- Egan, D.E. (1988). Individual differences in human-computer interaction. In M. Helander (Ed.), *Handbook of Human-Computer Interaction* (pp. 543–568). Amsterdam: North-Holland.
- Elkerton, J. (1988). Online aiding for human-computer interaction. In M. Helander (Ed.), *Handbook of Human-Computer Interaction* (pp. 345–364). Amsterdam: North-Holland.
- Ellis, M., Greer, K., Placeway, P., & Zachariassen, R. (1987, March 10). *TCSH - C shell with filename completion and command line editing* (UNIX Programmer's Manual (revised U of T, previous revisions from Fairchild, HP Labs., and OSU IRCC)).
- Engel, F.L., Andriessen, J.J., & Schmitz, H.J.R. (1983). What, where and whence: Means for improving electronic data access. *International Journal of Man-Machine Studies*, 18(2), 145–160.
- Ericsson, K.A. & Simon, H.A. (1980, May). Verbal reports as data. *Psychological Review*, 87(3), 215–251.
- Farr, M. (1989, Winter). MPW V.3.0: A tool for all reasons. In *APDAlog* (pp. 8–10). Cupertino, Ca.: Apple Computer, Inc.
- Feiner, S., Nagy, S., & Dam, A. van (1982, January). An experimental system for creating and presenting interactive graphical documents. *ACM Transactions on Graphics*, 1(1), 59–77.
- Fitter, M. (1979, May). Towards more “natural” interactive systems. *International Journal of Man-Machine Studies*, 11(3), 339–350.

- Fitts, P.M. (1954, June). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6), 381–391.
- Furnas, G.W. (1986, April 13–17). Generalized fisheye views. In M. Mantei & P. Orbeton (Eds.), *Proceedings of the CHI'86 Human Factors in Computer Systems* (pp. 16–23). New York: Association for Computing Machinery.
- Gasser, L. (1986, July). The integration of computing and routine work. *ACM Transactions on Office Information Systems*, 4(3), 205–225.
- Giroux, L. & Belleau, R. (1986). What's on the menu? The influence of menu content on the selection process. *Behaviour and Information Technology*, 5(2), 169–172.
- Goldberg, A. (1984). *Smalltalk-80: The Interactive Programming Environment*. Reading, Ma.: Addison-Wesley.
- Goodman, D. (1987, September). *The Complete HyperCard Handbook*. New York: Bantam Book.
- Gray, W.D., John, B.E., Stuart, R., Lawrence, D., & Atwood, M.E. (1990, August 27–31). GOMS meets the phone company: Analytic modeling applied to real-world problems. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Interact'90. Proceedings of the Third IFIP Conference on Human-Computer Interaction* (pp. 29–34). Amsterdam: North-Holland.
- Greenberg, S. & Witten, I.H. (1985, March). *Interactive End-User Creation of Workbench Hierarchies Within a Window System* (Technical Research Report No. 85/191/4). Calgary, Alberta, Canada: University of Calgary, Department of Computer Science.
- Greenberg, S. & Witten, I.H. (1988a, June). Directing the user interface: How people use command-based systems. In *Proceedings of the 3rd IFAC Conference on Man-Machine Systems* (pp. 299–305).
- Greenberg, S. & Witten, I.H. (1988b, May). How users repeat their actions on computers: Principles for design of history mechanisms. In E. Soloway, D. Frye, & S.B. Sheppard (Eds.), *Proceedings of the CHI'88 Human Factors in Computer Systems* (pp. 171–178). New York: Association for Computing Machinery.
- Greenberg, S. (1988a). *Using UNIX: Collected traces of 168 users* (Technical Research Report No. 88/333/45). Calgary, Alberta, Canada: University of Calgary, Department of Computer Science.
- Greenberg, S. (1988b). *Tool use, reuse, and organization in command-driven interfaces* (Research Report 88/336/48). Calgary, Alberta, Canada: University of Calgary, Department of Computer Science.
- Haerberli, P.E. (1986, June). A data-flow manager for an interactive programming environment. In *Proceedings of 1986 Summer USENIX Technical Conference*.
- Halbert, D.C. (1984, December). *Programming by Example* (Technical Report No. OSD-T8402). Palo Alto, Ca.: Xerox Office Systems.
- Hanson, S.J., Kraut, R.E., & Farber, J.M. (1984, January). Interface design and

- multivariate analysis of UNIX command use. *ACM Transactions on Office Information Systems*, 2(1), 42–57.
- Hayes, P., Ball, E., & Reddy, R. (1981, March). Breaking the man–machine communication barrier. *Computer*, 14(3), 19–30.
- HCR Corporation (1987, July). *Hi User's Guide (version 7.7)* (User Manual). Toronto, Ontario, Canada: HCR Corporation.
- Henderson, D.A., Jr. & Card, S.K. (1986a, July). The use of multiple virtual workspaces to reduce space contention in a window–based graphical user interface. *ACM Transactions on Graphics*, 5(3), 211–243.
- Holsti, N. (1989, April). A session editor with incremental execution functions. *Software–Practice and Experience*, 19(4), 329–350.
- Hutchins, E.L., Hollan, J.D., & Norman, D.A. (1986). Direct manipulation interfaces. In D.A. Norman & S.W. Draper (Eds.), *User–Centered System Design: New Perspectives on Human–Computer Interaction* (pp. 87–124). Hillsdale, NJ.: Lawrence Erlbaum Associates.
- John, B.E. (1988, May 6). *Contributions to Engineering Models of Human–Computer Interaction* (Technical Report). Pittsburgh, Pa.: Carnegie Mellon University, Department of Psychology.
- John, B.E., Rosenbloom, P.S., & Newell, A. (1985, April 14–17). A theory of stimulus–response compatibility applied to human–computer interaction. In L. Borman & B. Curtis (Eds.), *Proceedings of the CHI'85 Human Factors in Computer Systems* (pp. 213–219). New York: Association for Computing Machinery.
- Joy, W.N. (1980, November). *An Introduction to the C Shell*. UNIX Programmer's Manual (7th edition) 2c: Supplementary Documentation (Virtual–VAX II Version).
- Korn, D. (1983, July). KSH – A shell programming language. In *Proceedings of USENIX Association Software Tools Users Group (Summer Conference)* (pp. 191–202).
- Krishnamurthy, B. (1987, September). *Shells in an interactive system* (Technical Research Report No. CSD–TR–707). West Lafayette, In.: Purdue University, Computer Sciences Department.
- Kurlander, D. & Feiner, S. (1988, October 10–12). Editable Graphical Histories. In *Proceedings of the 1988 IEEE Workshop on Visual Languages* (pp. 127–134). Long Beach, Ca.: IEEE Computer Society Press.
- Larkin, J.H. & Simon, H.A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11, 1987.
- Larkin, J.H. (1989). Display–Based Problem Solving. In D. Klahr & K. Kotovsky (Eds.), *Complex Information Processing: The Impact of Herbert A. Simon* (pp. 319–342). Hillsdale, NJ.: Lawrence Erlbaum Associates, Publishers.
- Lau, F.C.M. & Asthana, A. (1984, March). Yet another history mechanism for command interpreters. *SIGPLAN Notices*, 19(3), 51–56.
- Lee, A. & Lochovsky, F.H. (1990a). Study of command usage in three UNIX

- command interpreters. In L. Berliquet & D. Berthelette (Eds.), *Work with Display Units 89: Selected Papers from the Second International Scientific Conference* (pp. 419–426). Amsterdam: North-Holland.
- Lee, A. & Lochovsky, F.H. (1990b, August). User's command line reference behaviour: Locality versus recency. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Interact'90. Proceedings of the Third IFIP Conference on Human-Computer Interaction* (pp. 121–128). Amsterdam: North-Holland.
- Lee, A. & Lochovsky, F.H. (1991, April 28 – May 2). *Visual search performance with an interaction history menu* (Interactive Poster). New Orleans, La.: CHI'91.
- Lee, E.S. & Raymond, D.R. (to appear). Menu-driven systems. In A. Kent & J.G. Williams (Eds.), *Encyclopedia of Microcomputers*. New York, NY: Marcel Dekker.
- Lee, A. (1990a). Issues in design of history tool for user support. In G. Cockton (Ed.), *Engineering for Human-Computer Interaction: Proceedings of the IFIP TC 2/WG 2.7 Working Conference on Engineering for Human-Computer Interaction* (pp. 273–289). Amsterdam: North-Holland.
- Lee, A. (1990b, May). Taxonomy of uses of interaction history. In S. MacKay & E.M. Kidd (Eds.), *Proceedings of Graphics Interface'90* (pp. 113–122). Toronto, Ontario, Canada: Canadian Information Processing Society.
- Lee, A. (1992a). User Support: Considerations, Features, and Issues. In R. Hartson & D. Hix (Eds.), *Advances in Human-Computer Interaction* (Vol. 3, pp. 190–234). Norwood, NJ.: Ablex.
- Lee, A. (1992b, May). *Accuracy of MHP/GOMS predictions for the task of issuing recurrent commands* (Short Talk). Monterey, Ca.: CHI'92.
- Lerch, F.J., Mantei, M.M., & Olson, J.R. (1989, April 30 – May 4). Skilled financial planning: The cost of translating ideas into action. In K. Bice & C. Lewis (Eds.), *Proceedings of the CHI'89 Human Factors in Computer Systems* (pp. 121–126). New York: Association for Computing Machinery.
- Lindsay, P. & Norman, D. (1977). *Human Information Processing: An Introduction to Psychology* (2nd ed.). New York: Academic Press.
- Linxi, C. & Habermann, A.N. (1986, August 13). *A history mechanism and undo/redo/reuse support in ALOE* (Technical Research Report No. CMU-CS-86-148). Pittsburgh, Pa.: Department of Computer Science, Carnegie Mellon University.
- MacGregor, J. & Lee, E. (1987a). Performance and preference in videotex menu retrieval: A review of the empirical literature. *Behaviour and Information Technology*, 6(1), 43–68.
- MacGregor, J. & Lee, E. (1987b, May). Menu search: Random or systematic?. *International Journal of Man-Machine Studies*, 26(5), 627–631.
- MacKenzie, I.S. (1989). A note on the information-theoretic basis for Fitts' Law. *Journal of Motor Behavior*, 21(3), 323–330.
- MacKenzie, I.S., Sellen, A., & Buxton, W. (1991, April 28 – May 2). A comparison

- of input devices in elemental pointing and dragging tasks. In S.P. Robertson, G.M. Olson, J.S. Olson (Ed.), *Proceedings of the CHI'91 Human Factors in Computer Systems* (pp. 161–166). New York: Association for Computing Machinery.
- MacLeod, M. & Tillson, P. (1990, August 27–31). Pull-down, holddown, or stay-down? A theoretical and empirical comparison of three menu designs. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Interact'90. Proceedings of the Third IFIP Conference on Human-Computer Interaction* (pp. 429–434). Amsterdam: North-Holland.
- Madison, A.W. & Batson, A.P. (1976, May). Characteristics of program localities. *Communications of the ACM*, 19(5), 285–294.
- Mantei, M.M. (1982). *Disorientation behaviour in person-computer interaction*. Ca.:University of Southern California, Annenberg School of Communication.
- Maulsby, D.L., Witten, I.H., & Kittlitz, K.A. (1989, August). Metamouse: Specifying graphic procedures by example. *Computer Graphics*, 23(3), 127–136.
- McMahon, M. (1987). *A practical system for managing mixed mode user interfaces* (working paper). Cambridge, Ma.: Symbolics, Inc..
- Mehlenbacher, B., Duffy, T.M., & Palmer, J. (1989). Finding information on a menu: Linking menu organization to the user's goals. *Human-Computer Interaction*, 4, 231–251.
- Miyata, Y. & Norman, D.A. (1986). Psychological issues in support of multiple activities. In D.A. Norman & S.W. Draper (Eds.), *User-Centered System Design: New Perspectives on Human-Computer Interaction* (pp. 265–284). Hillsdale, NJ.: Lawrence Erlbaum Associates.
- Myers, B.A. & Buxton, W. (1986, August). Creating highly-interactive and graphical user interfaces by demonstration. *Computer Graphics*, 20(4), 249–258.
- Myers, B.A. (1986, April 13–17). Visual programming, programming by example and program visualization: A taxonomy. In M. Mantei & P. Orbeton (Eds.), *Proceedings of the CHI'86 Human Factors in Computer Systems* (pp. 59–66). New York: Association for Computing Machinery.
- Neisser, U. (1964, June). Visual Search. *Scientific American*, 210(6), 94–101.
- Newell, A. & Card, S.K. (1985). The prospects for psychological science in human-computer interaction. *Human-Computer Interaction*, 1(3), 209–242.
- Nickerson, R.S. (1976, October 14–15). On conversational interaction with computers. In S. Treu (Ed.), *User-Oriented Design of Interactive Graphics System, Proceedings of the ACM/SIGGRAPH Workshop* (pp. 101–113). Pittsburgh, Pa.: Association for Computing Machinery.
- Nievergelt, J. & Weydert, J. (1980). Sites, modes, and trails: Telling the user of an interactive system where he is, what he can do, and how to get places. In R.A. Guedj, P. ten Hagen, F.R. Hopgood, H. Tucker, & D.A. Duce (Eds.), *Methodology of Interaction* (pp. 327–338). Amsterdam: North-Holland.
- Norman, D.A. & Shallice, T. (1986). Attention to action: Willed and automatic control of behavior. In R.J. Davidson, G.E. Schwartz, & D. Shapiro (Eds.),

- Consciousness and Self Regulation: Advances in Research and Theory* (Vol. 4, pp. 1–18). New York: Plenum Press.
- Norman, D.A. (1981, January). Categorization of action slips. *Psychological Review*, 88(1), 1–15.
- Norman, D.A. (1983, April). Design rules based on analyses of human error. *Communications of the ACM*, 26(4), 254–258.
- Norman, D.A. (1984, August 18–20). Cognitive engineering principles in the design of human–computer interfaces. In G. Salvendy (Ed.), *Human–Computer Interaction: Proceedings of the First USA–Japan Conference on Human–Computer Interaction: Vol. 1. Advances in Human Factors/Ergonomics* (pp. 11–16). Amsterdam: Elsevier.
- Norman, D.A. (1986). Cognitive engineering. In D.A. Norman & S.W. Draper (Eds.), *User–Centered System Design: New Perspectives on Human–Computer Interaction* (pp. 31–61). Hillsdale, NJ.: Lawrence Erlbaum Associates.
- Olsen, D.R. & Dance, J.R. (1988, January). Macros by Example in Graphical UIMS. *IEEE Computer Graphics and Applications*, 8(5), 68–78.
- Olson, J.R. & Olson, G.M. (1990). The growth of cognitive modeling in human–computer interaction since GOMS. *Human–Computer Interaction*, 5, 221–265.
- Paap, K.R. & Roske–Hofstrand, R.J. (1988). Design of Menus. In M. Helander (Ed.), *Handbook of Human–Computer Interaction* (pp. 205–235). Amsterdam: North–Holland.
- Potosnak, K.M., Hayes, P.J., Rosson, M.B., Schneider, M.L., & Whiteside, J.A. (1986, April 13–17). Panel: Classifying users: A hard look at some controversial issues. In M. Mantei & P. Orbeton (Eds.), *Proceedings of the CHI'86 Human Factors in Computer Systems* (pp. 84–88). New York: Association for Computing Machinery.
- Quinn, L. & Russell, D.M. (1986, April 13–17). Intelligent interfaces: User models and planners. In M. Mantei & P. Orbeton (Eds.), *Proceedings of the CHI'86 Human Factors in Computer Systems* (pp. 314–320). New York: Association for Computing Machinery.
- Rasmussen, J. (1983, May/June). Skills, rules, and knowledge: Signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3), 257–266.
- Reichman–Adar, R. (1986). Communication paradigms for a window system. In D.A. Norman & S.W. Draper (Eds.), *User–Centered System Design: New Perspectives on Human–Computer Interaction* (pp. 285–313). Hillsdale, NJ.: Lawrence Erlbaum Associates.
- Reisberg, D. (1987, July 16–18). External representations and the advantages of externalizing one's thoughts. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society* (pp. 281–293).
- Rich, E. (1983, March). Users are individuals: Individualizing user models. *International Journal of Man–Machine Studies*, 18(3), 199–214.
- Rissland, E.L. (1984, October). Ingredients of intelligent user interfaces.

- International Journal of Man-Machine Studies*, 21(4), 377–388.
- Rissland, E.L., Kolodner., & Waltz, (1989, May 31 – June 2). Case-based reasoning from DARPA: Machine learning program plan. In K. Hammond (Ed.), *Proceedings of a Workshop on Case-Based Reasoning* (pp. 1–14). San Mateo, Ca.: Morgan Kaufmann Publishers Inc..
- Ross, B.H. (1989, May 31 – June 2). Some psychological results on case-based reasoning. In K. Hammond (Ed.), *Proceedings of a Workshop on Case-Based Reasoning* (pp. 1–14). San Mateo, Ca.: Morgan Kaufmann Publishers Inc.
- Schonpflug, W. (1988). Retrieving texts from an external store: The effects of an explanatory context and of semantic fit between text and address. *Psychological Research*, 50(1), 19–27.
- Scofield, J. (1981, November). *Editing as a Paradigm for User Interaction A Thesis Proposal* (Technical Report No. 81–11–01). Seattle, Wa.: University of Washington, Department of Computer Science.
- Self, J.A. (1974, March). Student models in computer-aided instruction. *International Journal of Man-Machine Studies*, 6(2), 261–276.
- Sleeman, D. (1985, July). UMFE: A user modelling front-end subsystem. *International Journal of Man-Machine Studies*, 23(1), 71–88.
- Stallman, R. M. (1981, Spring/Summer). EMACS – The extensible, customizable self-documenting display editor. *SIGOA Newsletter*, 2(1–2), 147–156.
- Stearns, G.R. (1989, August). Agents and the HP NewWave application program interface. *Hewlett-Packard Journal*, 32–37.
- Stephenson, C.J. (1973). On the structure and control of commands. *SIGOPS: Operating System Review*, 7(4), 22–26.
- Suchman, L.A. (1983, October). Office procedure as practical action: Models of work and system design. *ACM Transactions on Office Information Systems*, 1(4), 320–328.
- Suchman, L.A. (1987). *Plans and situated actions: The problem of human-machine communication*. Cambridge, U.K.:Cambridge University Press.
- SUN Microsystems, Inc. (1986, February 17). *Windows and window based tools: Beginner's guide*. SUN Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043.
- Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12, 257–285.
- Sweller, J., Mawer, R.F., & Howe, W. (1982, Fall). Consequences of history-cued and means-end strategies in problem-solving. *American Journal of Psychology*, 95(3), 455–483.
- Taylor, M. (1988a, February). Layered protocols for computer-human dialogue. I: Principles. *International Journal of Man-Machine Studies*, 28(2), 175–218.
- Taylor, M. (1988b, March). Layered protocols for computer-human dialogue. I: Some practical issues. *International Journal of Man-Machine Studies*, 28(3), 219–257.

- Teitelman, W. & Masinter, L. (1981, April). The interlisp programming environment. *Computer*, 39–50.
- Teitelman, W. (1977, August 22–25). A display oriented programmer's assistant. In *Proceedings of IJCAI-77, 5th International Joint Conference on Artificial Intelligence* (Vol. 2, pp. 905–915).
- Teitelman, W. (1985, March). A tour through cedar. *IEEE Transactions on Software Engineering*, SE-11(3), 285–302.
- Thimbleby, H. (1980, October). Dialogue determination. *International Journal of Man-Machine Studies*, 13(3), 295–304.
- Tyler, S.W. & Treu, S. (1986, October 6–8). Adaptive interface design: A symmetric model and a knowledge-based implementation. *SIGOIS Bulletin*, 7(2–3), 53–60.
- Uejio, J.Y., Blattner, M.M., Schultz, E.E., & Ishikawa, M. (1991, January 8–11). A structured history for command recall. In *Proceedings of the 24th Annual Hawaii International Conference on System Sciences* (Vol. II, pp. 494–507).
- Vitter, J.S. (1984, October). US&R: A new framework for redoing. *IEEE Software*, 1(4), 39–52.
- Whitby, M. (1986, July). See MAC run. *MacUser*, 1(10), 38–42.
- Winograd, T. (1973, Winter). Breaking the complexity barrier (again). *SIGIR Forum*, IX(3), 13–30.
- Witten, I.H., Cleary, J.G., & Darragh, J.J. (1983, April). *The Reactive Keyboard: A new technology for text entry* (Technical Research Report No. 83/121/10). Calgary, Alberta, Canada: University of Calgary, Department of Computer Science.
- Witten, I.A., Greenberg, S.A., & Cleary, J. (1983, May 9–13). Personalizeable directories: A case study in automatic user modelling. In M. Wein (Ed.), *Proceedings of Graphics Interface'83* (pp. 183–189). Toronto, Ontario, Canada: Canadian Information Processing Society.
- Young, M., Taylor, R.N., & Troup, D.B. (1988, June). Software environment architectures and user interface facilities. *IEEE Transactions on Software Engineering*, 14(6), 697–708.