ED 362 212 IR 016 368

AUTHOR	Williams, Michael D.; Dodge, Bernard J.
TITLE	Tracking and Analyzing Learner-Computer Interaction.
PUB DATE	Jan 93
NOTE	16p.; In: Proceedings of Selected Research and Development Presentations at the Convention of the Association for Educational Communications and Technology Sponsored by the Research and Theory Division (15th, New Orleans, Louisiana, January 13-17, 1993); see IR 016 300.
PUB TYPE	Reports - Evaluative/Feasibility (142) -- Speeches/Conference Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	College Students; *Computer Assisted Instruction; *Computer Software; Data Analysis; Error of Measurement; Higher Education; Hypermedia; Interaction; Learning Strategies; *Man Machine Systems; *Measurement Techniques
IDENTIFIERS	*Audit Trails; *HyperCard; San Diego State University CA

ABSTRACT

Some specific computer-based tools for collecting and examining audit trails, i.e., data that describe a learner's path through a computer-based instruction (CBI) lesson, are detailed. Data analysis and measurement issues related to audit trails are also discussed. A particular set of embedded computer-based tools currently being used at San Diego State University (California) is described. These tools assess the what, where, and when of learner actions. The examples are developed in HyperCard. Researchers may attempt to interpret student responses by looking at the audit trail output file, but are probably better served by replaying the lesson to experience CBI as the learner did. Data from large numbers of learners may be aggregated for analysis. Issues of measurement error and concerns in dealing with aggregated data are reviewed. Audit trails can provide a great deal of understanding of how students experience CBI. Three figures illustrate the discussion. An appendix presents some scripts for audit trail handlers in HyperCard. (SLD)

ED 362 212

Title:

Tracking and Analyzing Learner-Computer Interaction

Authors:

Michael D. Williams
Bernard J. Dodge

IR016368

Introduction

The last twenty-five or so years have shown an explosion of research and development in computer-assisted and computer-managed learning environments. Academics investigating the nature of the learner-computer interaction, as well as software producers interested in developing higher quality, response-sensitive computer-based learning tools, have been seeking more and better means of collecting, managing, and analyzing the particular responses students make during their time at the computer keyboard.

When a learner interacts with instructional software, a complex set of processes takes place. Capturing this rich data for study is important if we are to advance the art and science of CBI design. But as the research questions have become more complex, there has also come a proliferation of options for data collection and analysis in these environments (Johnson, 1982).

The reasons for wishing to monitor student responses during instruction are myriad. Misanchuk and Schwier (1992) discuss four categories of reasons for wishing to collect data about online learner responses. The first category is for *formative evaluation in instructional design.* For example, software developers in an *alpha* testing phase of product creation are often interested to discover which features of the software could be streamlined or eliminated. Thus, for example, they may wish to trace the frequency or occasions when students access "help" screens or, maybe, how often students return to main menus.

A second category relates to *basic research in instructional design.* For example, basic researchers such as Clark (1984) suggest the need to try to determine precisely what students are thinking as they progress through computer-mediated learning systems in order to produce more efficient or effective instructional designs.

A third reason discussed by Misanchuk and Schwier (1992) is to collect such data for *usage audits for unstructured or public environments.* The purpose here is to unobtrusively peer "over the shoulders of of groups of users to determine how they a[illegible] traversing the interactive media package." This use of online response data combines characteristics of both of the first two reasons mentioned above.

Last, learner-computer interaction data might be used for *counselling and advising* purposes. That is, data from such interactions can be collected into databanks and analyzed with the intention of informing the learner of their relative skills, knowledge, learning styles, and so on. The idea is to involve the learner as a full partner in instructional decision-making.

But whether the investigator is a software developer concerned with the formative evaluation or the rapid prototyping of a new product, or is a researcher exploring the whys and wherefores of student responses during CBI, there is a clear need to have a toolkit of data collection and analysis techniques which can provide an accurate and useful record of student navigation routes.

This paper builds on the work of Misanchuk and Schwier (1992) by detailing some specific computer-based tools for collecting and examining what they call "audit trails," that is, data which describe a learner's path through a CBI lesson. (Incidentally, the term "audit trail" was also found in a remarkable early paper by Grubb, 1968, who presents a computer-based instructional system which in many respects presages the current hypermedia systems.) Additionally, data analysis and measurement issues relating to the use of data from these audit trails are also discussed.

Essential User Information to be Captured

If we are to track a learner's path through a piece of software, what kinds of data are important? In a graphic user interface (GUI) environment, the kind used most often today, there are three major categories of useful data which determine the learner's path through a computer experience:

1. ...the identity of each user interaction with the software. This information answers the question, "*What action* did the student take?" and may include:
   - mouse clicks on buttons, fields and cards
   - responses typed into the keyboard
   - menus pulled down

2. ...the specific screen location of each mouse event. This data answers the question, "*Where* did the action take place." and includes:
   - where the mouse was clicked
   - where the cursor was when the mouse button was released

3. ...the time at which each learner interaction occurred. From this data can be calculated any specific instance of what is sometimes called "latency," that is, the time spent between actions of the learner. The time taken to understand a screen or to answer a question can be used to study the learner's cognitive processing (e.g., see the work of Tennyson and associates).

In journalistic terms, these *What*, *Where* and *When* of learner actions will also be important to track in the new pen-based interfaces as they emerge. A few years from now, when voice-based interfaces become more common, the *What* and *When* factors will continue to be of interest to researchers and designers while the *Where* will become less relevant.

What follows is an in-depth presentation of a particular set of embedded computer-based tools currently being used at San Diego State University which assess the *What*, *Where* and *When* data types. Although the programming statements are idiosyncratic to the particular software used, the programming constructs represented are generalizable to other platforms and authoring systems, as well.

Structure of HyperCard

The examples that follow are developed in HyperCard, a tool-building environment for the Macintosh computer. Programming of a similar type could be done in any graphic-oriented environment on any platform, including Authorware, Plus, Director, ToolBook, and SuperCard. To clarify the descriptions of programming logic, some explanation of HyperCard's structure may be useful.

HyperCard is used to create *stacks* which are collections of individual *cards*. Cards can share common backgrounds and can contain *buttons* (active areas of the screen which can respond to input from the mouse) or *fields* (areas of the screen holding text, which can also respond to mouse or keyboard input). Stacks, cards, backgrounds, buttons and fields are collectively known as *objects*, and any object can contain a *script* or program consisting of one or more *handlers*. Each handler handles events as they occur, such as the click of a mouse within the boundaries of a button, or the pressing of the Enter key within a field.

Programming in HyperCard is oriented around these objects. Unlike some environments in which the intelligence is contained in a single program listed all in one place, a HyperCard stack can contain many programs in the form of handlers within individual buttons, fields and cards. The handlers for tracking learner actions are described in the next section.

Explanation of the Handlers

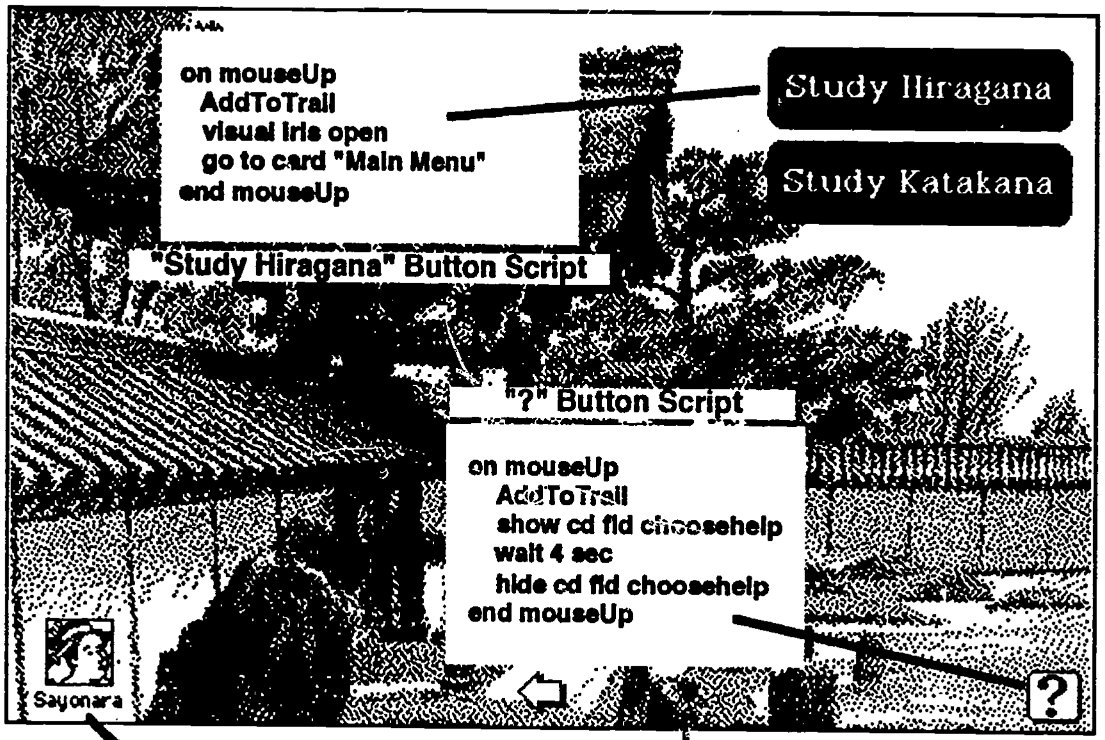
There are four major handlers needed to track learner actions in a HyperCard stack:

- *StartTrail* - initializes the trail variable which stores each action
- *AddToTrail* - adds a specific event to the trail
- *SaveTrailFile* - saves the trail variable to disk
- *ShowTrail* - shows all mouseclicks accumulated in a given trail file

Each of these is described in more detail below. The ShowTrail handler is discussed within the section on analysis options.

StartTrail. HyperCard makes use of global variables to store information while a stack is executing. Global variables retain their values anywhere in a stack, even across stacks, until the user quits the HyperCard application. Because the variable exists in random access memory, writing to it is quick enough to be practically undetectable. This allows the program to track user actions without slowing down the stack noticeably. In this instance, the startTrail handler sets up a global variable called "Trail". This variable contains separate lines of data for each learner action, and begins with a header consisting of bullets (•) to mark the beginning of a new run of the stack. This header is followed by a line containing the starting time and date on which the stack was run.

AddToTrail. Once the Trail variable has been set up, this handler stores data about each learner action. It is called up by including the name of the handler in the script of every button, field, and background in the stack. The illustration below shows how this might be done on a sample card:



"Study Hiragana" Button Script

```
on mouseUp
  AddToTrail
  visual iris open
  go to card "Main Menu"
end mouseUp
```

"?" Button Script

```
on mouseUp
  AddToTrail
  show cd fld choosehelp
  wait 4 sec
  hide cd fld choosehelp
end mouseUp
```

```
on mouseUp
  AddToTrail
  visual checkerboard slow
  go cd credits
end mouseUp
```

"Sayonara" Button Script

```
on mouseUp
  AddToTrail
end mouseUp
```

Background Script

SaveTrailFile. When the stack is closed, this handler takes the Trail variable from memory, adds the end time and total elapsed time, and stores it in a text file on disk.

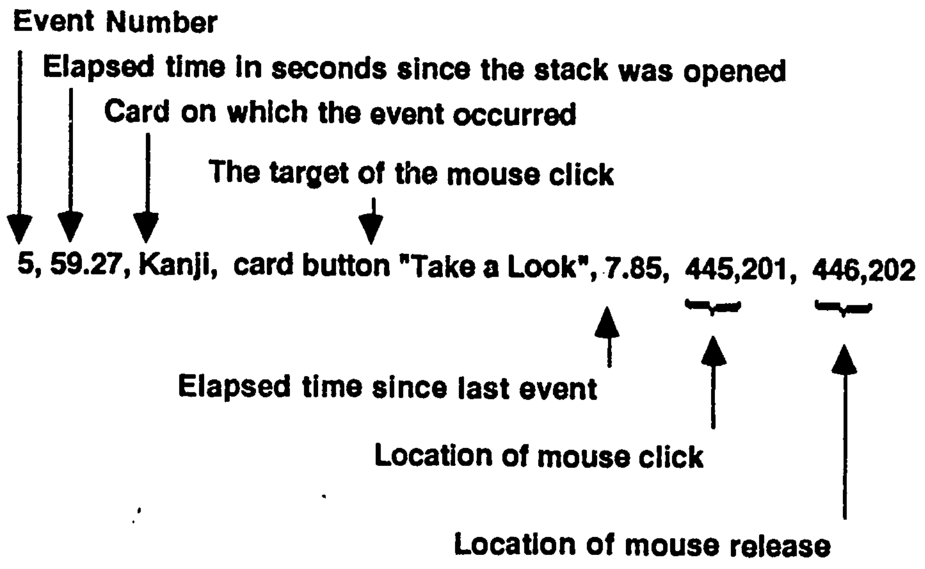
Sample Audit Trail

All of these handlers work together to create a record of learner actions stored on disk for later analysis. Here is a short sample trail:

```
••••••
Stack opened at 8:25:25 AM on Saturday, January 9, 1993

1,29.53,card id 109271,card button "Next",29.53,255,311,259,313,
2,33.18,Choice,card button "Study Hiragana",3.65,397,39,397,39,
3,42.53,Main Menu,card button "     Historical Stuff",9.35,394,59,397,62,
4,51.42,Historical,card button "Take a Look",8.88,447,202,447,204,
5,59.27,Kanji,card button "Take a Look",7.85,445,201,446,202,
6,62.33,Historical,bkgnd button "Menu",3.07,43,283,45,287,
7,65.38,Main Menu,card button "   Outta Here",3.05,371,239,371,239,
8,68.35,Choice,card button "Sayonara",2.97,40,305,54,300,
9,90.77,Last Card,card button "Quit",22.42,371,203,370,203,
10.00,90.90,Last Card,card button "Quit",0.13,371,203,363,203,Menu item: quit hypercard
Stack closed at 8:26:56 AM
Total elapsed time = 91.20 seconds
ΩΩΩΩΩΩ
```

This trail represents a very short run of the stack in which the learner passed through 6 different cards and clicked on 9 buttons to navigate among them. Each mouse click, menu operation, or keyboard input is stored as a single line in the trail, with each part of the data separated by commas.


Event Number
Elapsed time in seconds since the stack was opened
Card on which the event occurred
The target of the mouse click
5, 59.27, Kanji, card button "Take a Look", 7.85, 445,201, 446,202
Elapsed time since last event
Location of mouse click
Location of mouse release

Analyses of Audit Trails

The investigator has many decisions to make about what to do with the data from audit trails of student responses. This section will suggest methods which can help to make sense of the often times overwhelming volume of data generated.

As can be seen from the previous section on how to collect audit trails, if the investigator wishes to collect enough information to completely determine each student's path, actions, and time spent, the number of data points generated from a typical computer lesson can quickly run into the hundreds or thousands for just *one* student! Misanchuk and Schwier (1992), too, point out, audit trail data collected from multimedia/ hypermedia lesson structures can be extremely difficult to make sense of. This is because data for individual students are voluminous and data between students do not have a consistent or parallel format amenable to standard spreadsheet-oriented data analysis tools (including most commercial statistical packages, such as SPSS and SAS). Rarely is the investigator able to use, without convoluted data transformations, such standard statistical tools as repeated measures or time series analyses. Too often the volume of variables or the frequency of contingent missing data (that is, not all students experience the same instructional events) prevent adequate statistical interpretation of the data.

As it turns out, there is no "right" way to interpret such data. In general, though, there is one trade-off the investigator must make early in the data analysis process which will guide subsequent analyses: analyzing the intact data in its original and complete format, or analyzing the data which has been aggregated into summary, averaged, or otherwise collapsed variables. Finally, some pertinent measurement issues will be discussed.

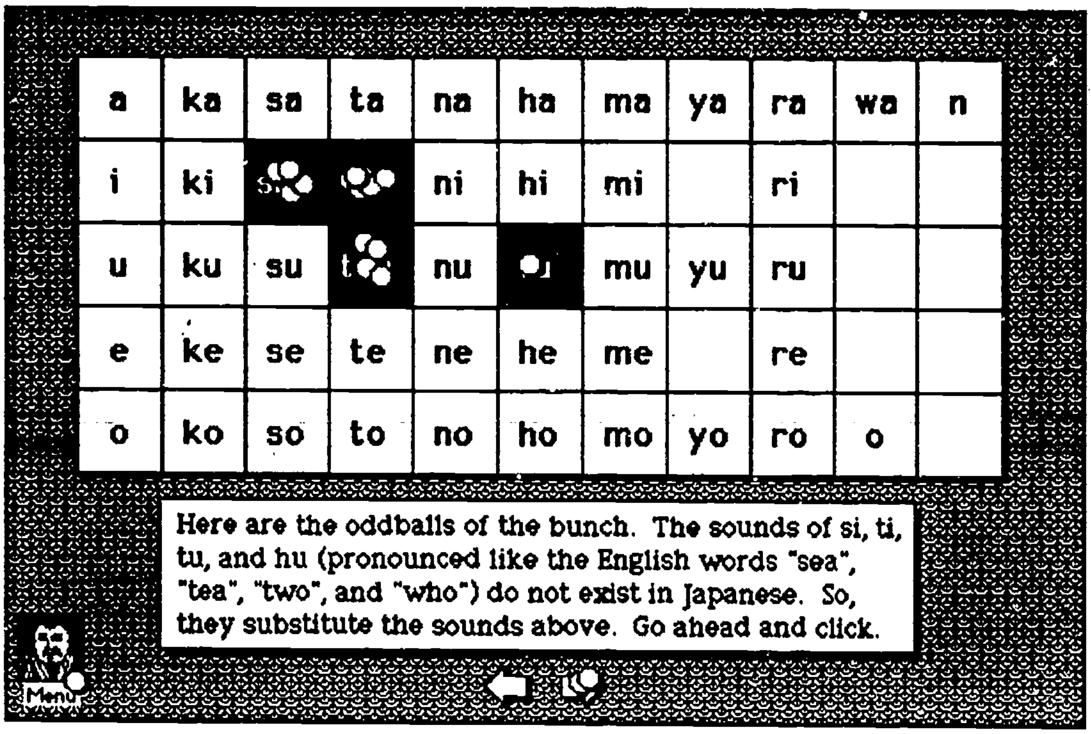
Qualitative analyses of original (non-aggregated) data

One option for investigators is to not even attempt to aggregate the data into reduced sets of records and variables. Indeed, there are many instances when investigators might wish to forego statistical analysis of audit trail data altogether. For example, instructional developers may wish to look at the data record from a manageably small sample of individual students to boost the validity of a formative evaluation. (A small *n* is usually required for such analyses because there are seldom enough resources available to conduct analyses on a large number of students.)

Additionally, researchers may want to adopt qualitative methods to examine each student's path up close. The idea (though far from simple or easy) is to search a fairly small number of audit trails for clues and patterns which might elucidate some phenomenon of interest connected with the lesson or the student. For example, this approach could unobtrusively give researchers valuable insight into a student's thought processes as they proceed through some lesson (e.g., Robson, Steward, & Whitfield, 1988). The challenge, of course, is maintain methodological rigor so that interpretations of individual audit trail data will be reasonably unbiased, and that findings can be validly generalized beyond the small sample involved. Many good sources exist which present techniques for qualitative data analysis of data such as this, so such techniques will not be covered here.

Rather than attempting to interpret student responses by simply staring at the audit trail output file, investigators are perhaps better served by, if possible, "replaying" the lesson so to experience the CBI just as the learner did. The following example illustrates just such an audit trail playback routine.

Illustration. The ShowTrail handler was developed to represent visually the history of mouse clicks recorded for one or more users of a stack. It takes a trial file, parses it, and places a round button on the location of every mouse click stored.



a	ka	sa	ta	na	ha	ma	ya	ra	wa	n
i	ki	[illegible]	[illegible]	ni	hi	mi		ri		
u	ku	su	[illegible]	nu	[illegible]	mu	yu	ru		
e	ke	se	te	ne	he	me		re		
o	ko	so	to	no	ho	mo	yo	ro	o	

Here are the oddballs of the bunch. The sounds of si, ti, tu, and hu (pronounced like the English words "sea", "tea", "two", and "who") do not exist in Japanese. So, they substitute the sounds above. Go ahead and click.

This allows one to see several things at a glance:

- Which buttons were clicked most often, least often
- Which items on the screen were misinterpreted to be buttons and were clicked on with no result
- What *part* of a button was clicked on.

In the example above, each button triggers a sound file so that the learner can hear the pronunciation of a syllable. The sounds on the buttons on the left are harder for the American ear to distinguish, and were thus clicked more often than the button playing the "hu" sound. In addition one can see that one learner left the sequence at this point to return to the menu, while the others clicked the forward arrow to continue.

Quantitative analyses of aggregated data

When data trails are involved from large numbers of students, it may not be feasible for the investigator to closely examine each student's record. In these cases, the investigator might want to collapse data into simpler categories or variables for analysis. For example, the investigator might want to average the student's time spent over only a few types of events; or perhaps simply count the number of times the student returns to, say, a review section of a lesson. These kind of aggregated data are much more amenable to statistical analysis, in that only a few variables are created and examined, and there are substantial numbers of students with data to provide statistical power. The trade-off here is that, whenever data are collapsed, information is lost. That is, the *unique* profile of each student's set of responses becomes lost in the aggregation process, and with it some potentially useful insight.

(Certainly, there are many instances when such aggregated data might be interpreted qualitatively, as in the previous discussion. However, it is recommended here that aggregated data are most useful when analyzed statistically, on a sample large enough to provide adequate statistical power.)

Types of variables. When examining aggregated data, the investigator usually has an agenda of research questions which will presumably be addressed by the data. Operationalizing these questions, however, is not a trivial matter because of the wide range of experiences students typically experience during hypermedia CBI. That is, the interpretations we might give to certain data points from one student might be different from the interpretations we give to another student on those same data points. Ignoring the *span* of each student's experiences might result in invalid conclusions about what occurred during the lesson.

Once the specific variables are decided upon, the researcher needs to decide which data points are to be included in the aggregated cases and variables. For example, say there is an embedded quiz which all students must complete. It is a fairly simple matter to aggregate these data and analyze them according to the research questions of interest. The same applies to any type of data which is common to *all* students, and generally fall into a few common types of data: nominal (e.g., multiple choice, menu selections), counts or totals (e.g., number of times a card is visited or button is clicked, total elapsed time spent on a lesson), and averages (e.g., average time spent reading expository material across a number of such events).

However, there is one type of data researchers frequently wish to collect, but which present some serious problems for analysis. For example, say the researcher wants to see how many times students visit a specific type of help screen. Certainly, the number of times students would choose to access the help would be contingent upon how many opportunities they had to access help. If some students had the opportunity to access help only once or twice, how is their help-selection data to be compared with someone who had many more such help opportunities? Such contingent data present continual analysis problems for researchers (Misanchuk & Schwier, 1992). Some statisticians indicate that such proportions are seldom normally distributed and recommend a number of possible transformations to better represent such data (Cohen & Cohen, 1975, p. 254-259). Even so, the interpretability of such data is limited.

More generally, in any hypermedia environment, there are likely to be a great many data points which are in some way conditioned on the student's experiences previously encountered in the lesson. It is imperative the researcher not jump too quickly to aggregate such data without first carefully examining the range of experiences the student traversed earlier in the lesson. The old saw "garbage in - garbage out" definitely applies here.

Data organization and format. Audit trail data generated by routines such as the one presented in this paper do not readily conform to the usual data input requirements of statistical programs. Indeed, only the most powerful of such programs (e.g., SPSS) are able to take as input almost any kind of form of data. Even so, such data may still need to be "cleaned up" prior to analysis. Such mundane features as commas and quotation marks need to be in the proper format for input into the statistical program. For example, the sample audit trail presented earlier in this paper would need to be somewhat modified prior to statistical analysis using SPSS. The string variables and the embedded quotation marks would need to be brought into conformity with the requirements of the program's data input routines.

It is very likely that the statistical analyst will need to employ other routines, as well, (such as SPSS's INPUT PROGRAM, END INPUT PROGRAM routines) to rearrange the input data to a more acceptable format.

Some analysis techniques and problems. Of course, the particular statistical routines employed will be a function of the research questions asked by the investigator. However, given the wealth of data collected, a number of statistical techniques (in addition to the usual MANOVAs and repeated-measures analyses) become available and attractive due to the existence for each student of a rich data "stream":

- ...multi-way contingency table analysis: related to the problem of contingent data mentioned earlier, the investigator might wish to create for each learner a contingency table of, say, choices given opportunities; these tables can themselves are able to be analyzed across the entire sample.
- ...the continuous supply of data from the student over time might offer researchers chances to look for trend patterns over time. Techniques such as time series and non-linear regression might be useful statistical techniques for such analyses.
- ...occasionally, researchers are interested in classifying students by "type" or "style." The idea is to see if students tend to cluster into meaningful groups based upon their online tendencies collected in the audit trail. While fairly unused in educational circles, a number of statistical profiling and clustering routines exist to help this sorting task (e.g., Hartigan, 1975).

Typical problems encountered during such analysis include, for example, choosing the proper unit of analysis (learner? Or where more than one student sits at a single keyboard, should the computer be the correct unit of analysis?), and determining whether a within-subjects design is appropriate and desired, and if so, choosing the proper design and error terms in the model.

Measurement Error

As with any data collection system, errors can occur in the measures which limit the reliability and interpretability of the information gathered. Although in a computer-based data collection system the measurement errors are not of the type we usually think of when humans collect and record data (data entry mistakes, lack of double checking, etc.), there are still a number of quality control issues which must addressed in order to have confidence in the data. Some of these potential problems are of concern in all measurement settings; others should be considered mainly when preserving non-aggregated data for qualitative analysis purposes; still others are of primary concern when statistical analysis is to be applied to aggregated audit trail data. (In fact, it is curious to note that what could be labeled a source of measurement error when analyzing aggregated data can often be pinpointed when looking at the same data in its original non-aggregated form, that is, by examining each student's audit trail individually.)

General measurement concerns:

- *General disruptions.* Sometimes students kick out a power plug in the middle of a lesson, or a student receives a bad disk which crashes abruptly, or fire alarms sound. The investigator must decide how much incomplete data can be tolerated, and whether the disruption tainted the data collection so as to be invalid or otherwise unusable.

- *Accidents and carelessness.* Students occasionally press the key they did not want. Sometimes they realize their mistake; sometimes they don't. How the computer can provide a means for the student to check and/or modify their answer without being overly intrusive to the overall flow the lesson is a common challenge to researchers.

- *Offline behaviors:* Here the investigator is interested in what the student is doing with the computer besides pressing keys. For example, are students talking with each other or looking on each other's screens? Are students referring to job-aids, workbooks, or other offline materials? How interested do the students look? What is the nature of the interactions between students when working cooperatively? Most often, such offline behaviors are most validly assessed with unobtrusive video or audiotape, but can be assessed with live observers, as well. These types of data can be examined with traditional statistical techniques (e.g., Webb, 1984) or might be treated qualitatively.

Measurement concerns when analyzing aggregated data

- *Awkward means of responding.* If the student is unclear what to do to indicate their response or if it is awkward for them to do so, students might end up making many unintended mistakes or even giving up. Multiple or non-standard keystrokes required to make a simple choice, poorly labeled screen "buttons" or bad screen design distract students from the matter at hand to focus instead on just making the computer work correctly. Although awkward response mechanisms presents a source of measurement error when data are to be analyzed in aggregate, identifying such responses in *specific*, that is, non-aggregated audit trails might be of prime interest during a formative evaluation.

- *Student disinterest.* Otherwise known as, "I just want to get through this damn computer lesson as fast as possible..." Students might want to just hit any key to get through the lesson because they just don't care what happens. The investigator who unquestioningly trusts all data from the computer risks drawing unwarranted inferences about the data (Damarin & Damarin, 1983). Again, such a source of measurement error can often be readily identified when the data are examined on an individual, non-aggregated basis.

- *Question intrusiveness.* This is a thorny issue common to many educational studies. The investigator should be aware that the very act of collecting data might alter the phenomenon being studied. Occasionally, researchers will insert into a computer lesson so many non-critical questions (options, opinions, confirmations, confidences, etc.) that the lesson gets exhausting to go through. The students are in danger of feeling like, "Let's just get ON with it!" This intrusiveness is particularly evident in later stages of the lesson, as the student gets more and more irritated with having to answer seemingly irrelevant questions. Identification of this phenomenon, too, might best occur when examining individual audit trails, perhaps through the "playback" technique presented earlier.

Measurement concerns when analyzing individual, non-aggregated data

- *Individual differences in learners.* Say the investigator wants to look at time spent on the lesson and collects such data online. Such factors as the learner's typical reading speed or physical coordination level (i.e., their motor skills for handling the input devices) have the potential to provide false information about the time-on-task the typical student might take. Additionally, some students have more familiarity with computers (keyboards, cursors, control keys, etc.) than others. When analyzing aggregate data, these effects can be considered random, but when looking qualitatively at individual, non-aggregated data the investigator should be aware of the possibility that these individual differences might confound or dilute interpretation of the data. That is, it may be difficult to draw any valid conclusions about the performance of a lesson from only looking at a few students.

Summary

A method for collecting "audit trails" of data generated by students as they proceed through computer-based instruction was presented and discussed. Specific HyperCard routines were delineated which completely capture the student's online experience, and are able to "playback" the responses to the investigator.

Additionally, data analysis issues surrounding the use of audit trails were presented. The purposes of aggregating or not aggregating such data were compared, some statistical concerns were offered, and measurement issues potentially affecting the interpretability of the collected data were reviewed.

In general, it is expected that techniques and issues presented in this paper will go a long way to improving our understanding of how students experience computer-based instruction, and how we might improve our instructional designs to reflect these human factors.

Appendix: Scripts

All of the handlers that follow are to be installed in the stack script of the stack to be audited. Comments are included in each to make them self-documenting.

```
on openStack
  StartTrail
end openStack
```

```
on StartTrail
  ------------------------------------------------------------
  -- This handler sets up a global variable called Trail.
  -- It begins with a header with bullets to indicate the
  -- start of a new play of the stack, followed by a line
  -- showing the starting time and date.
  -- The third line is left blank, but stacks can be scripted
  -- to put the user's name, experimental condition, or other
  -- data here.
  ------------------------------------------------------------
  global startTime,trail,event,lastEventTime
  put "••••••" & return into trail
  put "Stack opened at" && the long time after trail
  put " on" && the long date & return after trail
  set numberFormat to "0.00"
  put the ticks/60 into startTime
  put 0 into event
  put startTime into lastEventTime
end StartTrail
```

```
on closeStack
  SaveTrailFile
end closeStack
```

```
on SaveTrailFile
  -- This writes the Trail global variable to disk

  ------------------------------------------------------------
  -- End the trail global with closing time and elapsed time
  ------------------------------------------------------------
  global startTime,trail
  put the ticks/60 into endTime
  set numberFormat to "0.00"
  put return & "Stack closed at" && the long time & return after trail
  put "Total elapsed time =" && endTime - startTime && "seconds" & return after trail
  put "ΩΩΩΩΩΩ" & return after trail

  ------------------------------------------------------------
  -- The next line specifies the volume and name of the trail file.
  -- By default, "TrailFile" will be created in the same folder
  -- as HyperCard itself. If you wanted to put it somewhere else
  -- you would put the full pathname into the variable TrailFile.
  -- For example, put "CirrusSys:Data:TrailFile" into TrailFile
  -- would define TrailFile as a file called 'TrailFile'
  -- in a folder called 'Data' on a drive named 'CirrusSys'
  ------------------------------------------------------------
  put "TrailFile" into TrailFile

  ------------------------------------------------------------
  -- Next, read to the end of the trail file, if it exists, so
  -- that this latest trail will be appended to it. If there is
  -- no trail file, this line creates one.
  ------------------------------------------------------------
  open file TrailFile
  repeat
    read from file TrailFile for 16384
    if it is empty then exit repeat
  end repeat
  write trail to file TrailFile
  close file TrailFile
end SaveTrailFile
```

```
on AddToTrail input
 --------------------------------------------------------------
 -- This handler stores data about mouseclicks in a line in a
 -- global called Trail along with the time at which they occurred
 -- and the target of the mouseclick.
 -- Item 1 = the event number, starting with 1
 -- Item 2 = the total elapsed time in seconds since the stack opened
 -- Item 3 = the target of the mouseclick (a field, button or card)
 -- Item 4 = the name of the card
 -- Item 5 = the elapsed time since the last mouseclick
 -- Item 6 & 7 = the coordinates of the mouseclick
 -- Item 8 & 9 = the coordinates where the mouse button was released
 -- Item 10 = Non-click events such as pulling down menus or
 --         user input to a field.  This is a parameter passed
 --         when necessary.  See the domenu handler, for example.
 --------------------------------------------------------------
 global startTime,trail,event,lastEventTime,DontAddToTrail
 put the ticks/60 into nowTime
 if DontAddToTrail = true then exit AddToTrail
 add 1 to event
 put event + 3 into trailLine
 put trunc(event) into item 1 of line trailLine of trail
 set numberFormat to "0.00"
 put nowTime - startTime into item 2 of line trailLine of trail
 put the short name of this card into item 3 of line trailLine of trail
 put the name of the target into item 4 of line trailLine of trail
 put nowTime - lastEventTime into item 5 of line trailLine of trail
 put the clickloc into item 6 to 7 of line trailLine of trail
 put the mouseloc into item 8 to 9 of line trailLine of trail
 put input into item 10 of line trailLine of trail
 put nowTime into lastEventTime
end AddToTrail
```

```
on domenu x
 AddToTrail "Menu item:" && x
 pass domenu
end domenu
```

```
on ShowTrail
 ---------------------------------  ------------------------
 -- This handler creates a representation of the usage patterns
 -- of a stack.  It parses a trail file containing data from
 -- one or more runs of a stack and marks the location of
 -- every mouseclick on each card.
 -- To execute this handler, call up the message box,
 -- type "ShowTrail", and press the return key.
 -- This should be done on a COPY of the stack.
 -- The first section creates a new button named Breadcrumb
 -- in the form of a small shadowed circle.
 ----------------------------------------------------------
 global DontAddToTrail
 put true into DontAddToTrail
 domenu new button
 get the id of cd btn "New Button"
 put it into x
 set the showName of cd btn id x to false
 set the name of cd btn id x to "Breadcrumb"
 set the rect of cd btn id x to 100,100,112,112
 select cd btn id x
 domenu "Cut Button"

 ----------------------------------------------------------
 -- Next we locate the file that captured the trail
 ----------------------------------------------------------
 answer file "Where is the trail file?"
 if it is not empty then
  put it into TrailFile
 else
  choose browse tool
  put false into DontAddToTrail
  exit ShowTrail
 end if
 open file TrailFile
 put "•" into TheLine
 set lockmessages to true
```

15

```
-------------------------------------------------------
-- Next, we go thru each line of the TrailFile, looking
-- for lines representing mouseclicks
-------------------------------------------------------
repeat until theLine is empty
  read from file TrailFile until return
  put it into theLine

  -------------------------------------------------------
  -- If it's one of the lines with date, "•", etc., skip it.
  -------------------------------------------------------
  if the number of items in theLine < 9 then next repeat


  -------------------------------------------------------
  -- Go to the card where the event occurred
  -------------------------------------------------------
  put item 3 of theLine into theCard
  if word 2 of theCard = "id" then
   do "go" && theCard
  else
   do "go card" && quote & theCard & quote
  end if

  -------------------------------------------------------
  -- Find the location of the mouseclick, ignoring pull down
  -- menus and inputs into a field, and paste the Breadcrumb
  -------------------------------------------------------
  put item 6 to 7 of theLine into theLocation
  put item 10 of theLine into nonClick
  if nonClick contains "Menu item:" or nonClick contains "Input:" then
    next repeat
  else
   domenu "Paste Button"
   put the number of cd btns into x
   set the loc of cd btn x to theLocation
  end if

 end repeat

 ----------------------
 -- Now tidy things up
 ----------------------
 close file TrailFile
 choose browse tool
 put false into DontAddToTrail
 play harpsichord
end showTrail
```