APPENDIX: BUILDING AN EVENT SEQUENCE FILE

David N. Grazman and Andrew H. Van de Ven

Introduction

This appendix takes the researcher, step-by-step, through the mechanics involved in building an event sequence file, from defining a qualitative datum to analyzing the temporal relationships in event sequence data. While each longitudinal study of organizational change processes is characterized by its own specific requirements, this discussion will utilize the CIP data to illustrate the steps. By design, this appendix leaves substantial room for interpretation and application. We do not intend for it to serve as a manual for any particular software program, but as a general guide to the steps involved in collecting, managing, and analyzing large sets of event data.

The methods described here relate to recording and analyzing event data (consisting of descriptions of actors, actions, outcomes, dates and sources) from real-time or archival longitudinal, qualitative research. We refer to the "researcher" as the person who participates in all steps of the research, including collecting data, designing the database, managing the data, and analysis. Although field studies may require the involvement of many data collectors, we find that the fewer the number of individuals involved in managing or changing the actual event database, the less likely major inconsistencies or errors arise in constructing the dataset. Keeping responsibility for entering and working with raw event data with as few individuals as possible is one way to ensure higher levels of data consistency.

Hardware and Software Requirements

Computers are a crucial factor in determining how easy or difficult data entry and manipulation tasks become. Most longitudinal field studies entail collecting enormous amounts of data often obtained over lengthy time periods. Therefore, it is important to use computers with sufficient RAM and hard drive storage space. Most software programs basic for this research require large amounts of RAM and are significantly slowed (or do not run at all) when adequate memory is not available.

To minimize data management errors, we suggest that data files be stored on a single computer and that one version of the file serve as a master. If more than one researcher is involved in entering or managing data, it becomes crucial to have a single data storage site so that the most up-to-date and complete data file will always be used. Research files grow quickly and data file backups are best made often and kept by one researcher separately

from the working master copy. Without these precautions, it is inevitable that changes made in one copy of the file often do not find their way to other copies and people begin to work with slightly different versions of the database, creating a problem that is difficult to pinpoint and even harder to correct.

We are using two commercially available software packages, R:Base© and RATS©, though these are by no means the only programs on the market that have the necessary capabilities for longitudinal research. We are currently evaluating newer, more powerful software packages. Whatever programs you choose to use, all must be able to store and read data files in unformated ASCII code in order to facilitate transfer of data from program to program. Fortunately, it is increasingly common for software packages to import and export data formated by other packages, and at some point this requirement may no longer be necessary.

R:Base© (Microrim, Inc.) is a Windows-compatible, relational database program that allows information to be entered via predesigned forms into data tables that can be sorted, indexed, reorganized, and accessed quickly and easily, with minimal training. Many of the procedures we cover are tailored toward R:Base but can be easily adapted to other database programs. At the time of our review of database software, we found that R:Base had a unique advantage of allowing unlimited length "note" fields. For qualitative event data, it is useful to have the freedom to enter as much information as necessary without worrying about hitting the end of the data field.

Regression Analysis of Time Series© (RATS, VAR Econometrics) is a statistical analysis program focused on time series analysis and the graphing of results. RATS is a relatively straightforward program to use, however, it takes familiarity with its syntax before one feels comfortable using the program and knowing its capabilities. We are currently examining other software programs that may be more well suited and powerful than RATS, as well as more user-friendly. We do not anticipate, however, that the fundamental issues involved in event data analysis would change if we were to use another package. As with R:Base, our example uses RATS because that is what we currently use. However, the principles relate to most time series analysis packages available.

Planning, Planning, Planning

Even with appropriate hardware and software, we are still *not* ready to begin entering data. The data management process described here is fairly complex and may involve many data files, many variables, and literally thou-

sands of observations. Careful planning at the outset helps ensure (though it does not guarantee) that problems down the road are few and that when they do arise, they can be corrected. Planning ahead also helps to maximize the effectiveness of data analysis; indeed, analyses that could have resulted in valuable findings may be impossible to carry out if data is coded or stored sloppily. The value of this appendix comes not only from the detailed instructions it provides, but also from pointing out the lessons we have learned over time from our own mistakes, many of which were completely preventable.

The process of building an event sequence data file is pictured in Figure 5.3. The following sections describe how to complete the research tasks outlined in Chapter 5 that relate to data collection, management, and analysis. The steps covered in this appendix do not necessarily correspond one-to-one with the sections in the discussion in chapter 5, but are presented in sequential order and are logically integrated with and connected to the contents of chapter 5. We will assume that the researcher has decided on a design, chosen an appropriate sample, and has obtained data. At this point, the researcher must prepare that data for further analysis. This process can be broken into five steps, which will be discussed in turn. Following this, we will describe a sixth step, conducting a time series analysis; theory and methods for this type of analysis will be outlined in chapter 8.

Step 1. Designing a Database File to Record and Manage Event Data

First, the research team must define a qualitative datum, enter raw data into incidents, and assess the reliability and validity of the incidents. Chapter 5 provides a conceptual overview of incidents and discusses the processes involved in validating meaningful incident records. However, before incidents can be recorded, a database file must be built to handle the raw event data. The design of the database file is the heart of the first step outlined in Figure 5.3.

Setting the Incident Format

As we observed in chapter 5, before data collection or file creation takes place, researchers must agree upon the components of an event for theoretical reasons. There is also a mechanical reason: Researchers involved in collecting the information that will be used to specify incidents must have in mind clear definitions of the minimum bits of information necessary to document events in a standard manner. This is true whether the informa-

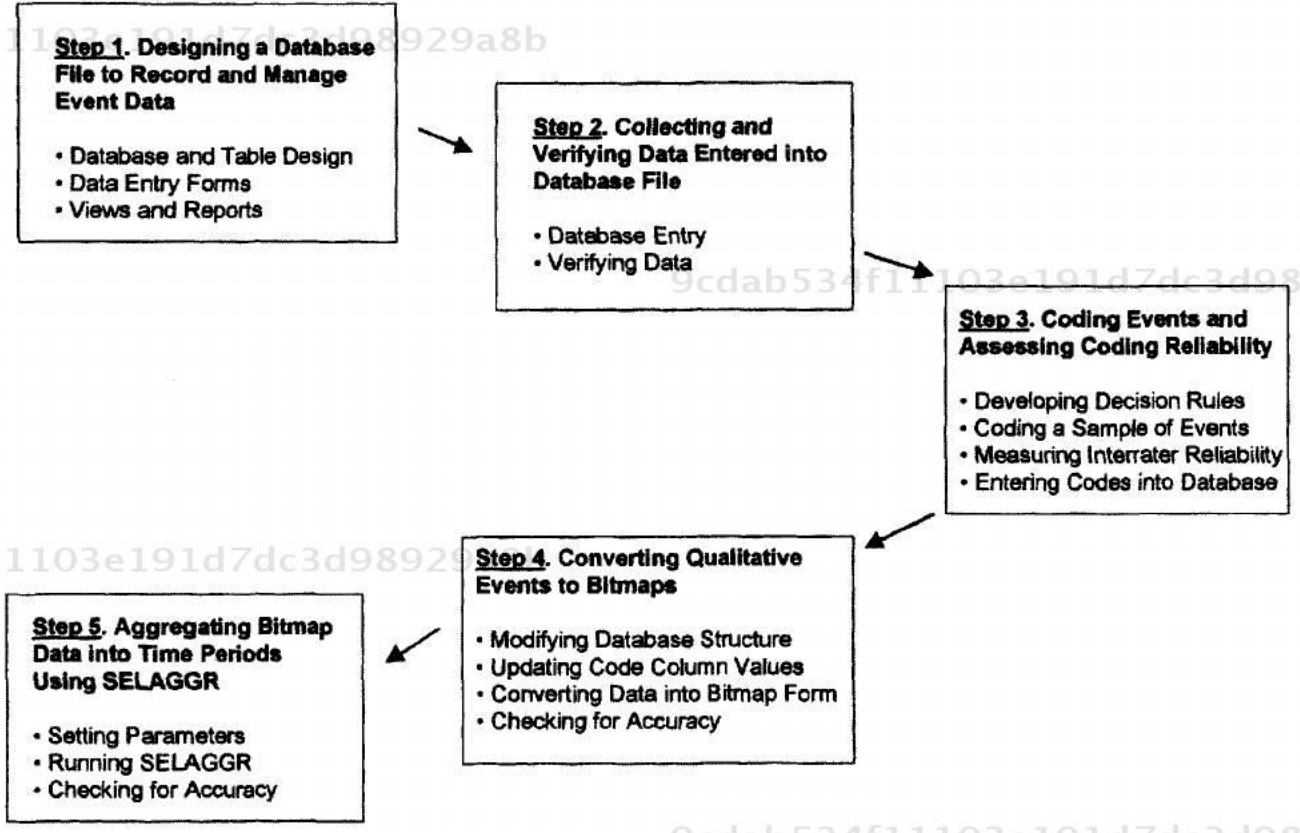

Figure 5.3 The process of building an event sequence data file

tion is gathered from archives or in the field. If the needs of the project are unclear, there is a danger that data will be incomplete. And without standardization it is hard to compare events or to understand how one event leads to another.

For example, for the CIP study, the decision rule used to specify an incident in the CIP study was that each incident should contain, at a minimum, the name of a primary actor (and secondary actor, if appropriate), a description of the action taken or change made, the date the action took place, and if known or discernible, the outcome of the action, why the action took place, and the source of the information. Agreement about these components prior to the initial database design saved time and effort and prevented having to go back to the raw events to change them later. Though it is possible to change data after it has been entered, it is a tedious task.

Database and Table Construction

Once a standard event format has been set, a researcher must design the database file where the data will be stored. This is a crucial step in the data management process because the design of the database determines how data are accessed for editing, analysis, and reporting. Various database programs have different structural capabilities. Our example uses R:Base©, though the principles covered relate to most current database programs.

A database file can consist of a number of tables with relational connections, meaning one data field can link together related entries in different tables. We have found that one table is adequate for many longitudinal cases, as long as the data coding scheme allows for sufficient complexity. Therefore, we do not go into detail here about the use of multiple data tables.

The researcher needs to create and define the table into which collected data will be entered. A table gives the data an organized structure and allows for access, viewing, and editing. Each row represents a single event and each column or "field" contains information relating to that event. Each column should represent the event components agreed upon earlier and be placed into the table in a logical order for manipulation. For example, if an event is composed of a date, an actor, and an action, it is easiest to place the date column to the left of the others, so that it is easy to see the date for each event when rows are chronologically sorted. Figure 5.4 shows a sample data table.

When defining columns (fields) in a table, each column's format must match the information that will be entered into it. Fields need to be defined

Days	Event	Observation	Source	Keywords	
01/01/77	Researchers in Los Angeles conduct the 1st cochlear implant in the US by implanting a limited number of patients using a single electrode device.	The event was published in W. F. House and K.. Berliner's "Cochlear Implants: Progress and Perspectives," Annals of Otiology & Rhinology, 1982, pp. 1-124.	ASHA, May, 1985	House, Academics	More Data Fields →
	↓ More Events				

Figure 5.4 Sample event data table from R:Base

as date fields, text fields, note fields, or integer fields. Field definition sets the format for the data and determines how that data can be used. Formats include the length of the data in the field. However, the length of note fields in R:Base© does not need to be specified and can be approximately 4,000 words. Unlike text fields, the content of note fields cannot be searched for key words or phrases. Allow for sufficient length in each text field because it is difficult to expand the size once data are entered. Events, or rows in the table, can be alphanumerically sorted using any field as a key. Once columns are defined, the table is saved as the central component of the database file.

Along with columns for incident components, we also suggest including a separate column to be used for code words (keywords) to allow for sorting and grouping of events on their coded dimensions. We discuss coding and keywords later in this appendix, but this section warrants their mention. With a simple coding scheme, a single keyword field (usually, a text field), is sufficient. Make it long enough to contain multiple codings. More complex coding schemes can often be simplified by adding more than one keyword field. For example, if three general categories of codes are used (for changes in transactions, people, or ideas), with subcodes for each category, adding

three uniquely named keyword fields may be the most useful way to handle coding and grouping. Different coding schemes are addressed in more detail below.

Data Entry Forms

Although data can be entered directly into the table, we suggest using customized data entry forms for quicker and more accurate data entry. Directly linked with the data table, data entry forms are designed to match column names and data types. Data entry forms give researchers an easier way to input data because all fields related to a unique incident are visible at one time. Figure 5.5 shows an example of a data entry form. The field order in which data are entered can be programmed in the data entry forms and can be designated with efficient entry in mind. Data entry forms serve as templates for incoming event data, and while they can be adjusted to reflect new linkages with a table, we suggest limiting modifications in order to keep entries as consistent as possible.

Data entry forms are also useful as printed copies that can be distributed to data collectors to fill in all fields on the sheet before submitting it for entry. By doing this, not only is the researcher more likely to get complete events, but also event information will be compatible with the entry form for the file itself, greatly speeding up and ensuring the integrity of the data entry process.

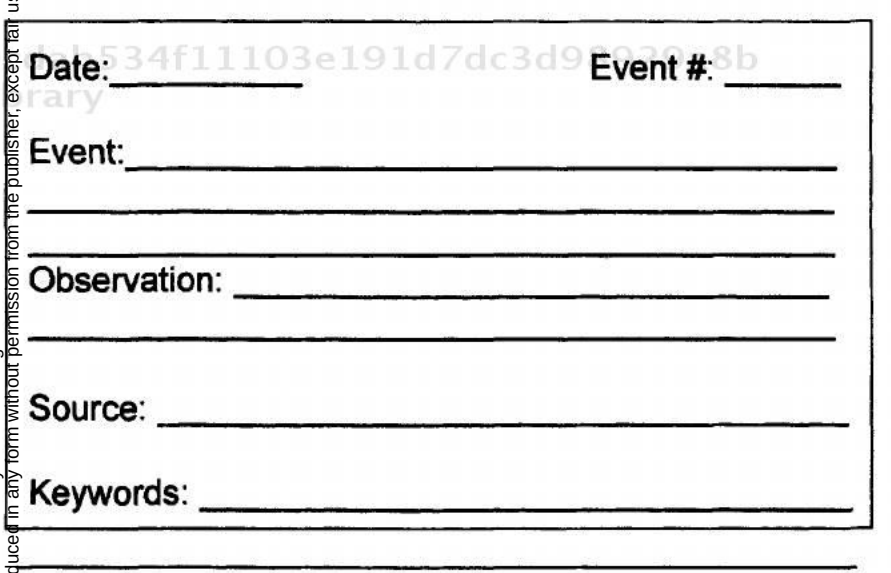

Figure 5.5 Sample data entry form

Views and Reports

Views and Reports offer tools with which researchers can systematically view, manage, or print incidents in a data file. Though these steps of database construction may be better discussed or understood following an introduction to coding, it is an important element of the file's infrastructure and should, at least, be considered throughout the planning process.

Because we advocate using one data table, all incidents are entered into a data file for each individual case. Although one table makes data management easier, it makes it more difficult to look directly for certain classes of events separately. R:Base offers a filterlike option called a "View" that allows a predetermined subset of events, based upon codes, dates, or other criteria, to be viewed independently from the entire set of events. To construct a view, the researcher must gain access to the data table itself, then follow the same procedures as called for by a database query. R:Base's on-line help command walks an operator through these steps. Views can be continually and easily updated and changed.

Predesigned reports allow the researcher to create text or screen output of the entire set or any particular subset of events sorted by date or other criteria. Reports can be formated to include any and all columns in the data table. In R:Base, the "Reports" menu contains an option to "Create/Modify," from which a report can be built and stored for future use. At the very least, we recommend setting up an initial report that prints all fields basic to each event. This report is crucial because it will be used in the iterative process of verifying data, coding events, and determining the reliability of the coding scheme. Table 5.4 from chapter 5 shows an example of a printed summary report of events.

Step 2. Collecting and Verifying Data Entered into a Database File

Once the structure of a data file has been constructed and stored, researchers can begin entering real-time and archival data. The data collection process can be tedious as well as exciting; most longitudinal projects will consist of periods of both. As always, planning and organization are key to managing the data collection process from beginning to end. Researchers should strive to enter complete event information; however, if only partial information is available for an event, it is best to enter partial data and return to update or correct it at a later time.

Data Entry

Once a database structure is defined, researchers can begin entering the event data that will serve as the raw materials upon which keyword coding and time series analysis are based. A general rule for data entry in the case of longitudinal, real-time studies is to enter the information as soon as possible after it is collected. Event data piles up very quickly and the researcher who waits to enter data after too much has been collected can become overwhelmed. Moreover, the sooner information is entered into a database, the more likely it is to be accurate. If it is inaccurate, the sooner it is noticed, the more likely the researcher will be able to track down the original source of the information and correct the incident record.

In the case of archival studies, systematic searches of databases, company documents, library records, newspapers and magazines, academic journals, and other sources should be conducted at regular intervals over the time span covered by the research so that as much relevant event information as possible can be gathered and entered into the database. Interviews, conversations, and correspondence should be recorded, transcribed, and entered into the database as soon as possible after they occur to minimize the risk of forgetting nuances of the interview or other important issues relating to the conveyed information. Expediency in entering data is important to prevent misstatements or errors from becoming part of the permanent data file.

Original source documents should be recorded and stored in an orderly way to ensure that if discrepancies in the data exist, researchers can reexamine the original documentation to verify or correct event information. Files should be kept by the researcher in a safe and organized way and should be easily accessible. One useful strategy for keeping data sources organized is to use uniform classifications in the source field in the event table. This way, events can be quickly and easily linked back to their original source.

Verifying Data

Data entered into the database must be repeatedly checked for completeness and accuracy. As was the case for data entry, verifying data is an easier task if it is done while it is being entered or shortly thereafter, instead of waiting until the entire data file is complete and then returning to check for accuracy. In general, there are two types of verification that can be done in order to minimize the possibility of errors in the events that are included as part of the database. One involves checking original documentation, and the other involves printing events in a report and sharing it with informants who can verify its accuracy.

First, all of the categories that are chosen to make up the event construct—dates, actors, outcomes, etc.—should be verified. If researchers uncover references to the same event from two or more different sources, each account should be entered so that any biases due to a data collector or source are minimized. Later, researchers can decide which account is more accurate or if multiple accounts of a single event need to be included.

Second, data accuracy can also be judged by organizational informants, participants, or others familiar with the events being observed. Input from these informed sources is particularly useful in detecting perceptions and unrecorded events and can, itself, serve as an important source of additional information. In conducting this type of study we have found that organizational informants are very effective in pinpointing errors in the data, elaborating on the data already collected, and leading us to new sources of information that we did not previously know existed.

Step 3. Coding Events and Determining Code Reliability

Once event data have been entered and verified, we can move to tasks three and four described in chapter 2: coding incidents into event constructs and assessing the reliability and validity of our coding scheme. These tasks comprise the first steps in the analysis of the accumulated data. Figure 5.6 summarizes the tasks involved in translating a database of incident listings into a reliable set of coded, theoretically meaningful events.

Developing and Refining Decision Rules

An effective coding scheme accomplishes two objectives. First, it captures the theoretically important dimensions of the phenomena that the researcher is interested in tracking. Second, it provides a guide to the classification and categorization of incidents in the database.

To accomplish the first objective, the researcher should be clear about the theoretical grounding for his or her concepts. This requires clear definitions of event concepts, constructs and indicators, and decision rules for achieving high construct validity as concepts move up and down the ladder of abstraction. The development of decision rules should specify operational steps for classifying incidents into event constructs the researcher has chosen to address a particular research question.

For the second objective, the evolving decision rules should be frequently checked against the data to ensure they are meaningful and complete guides for classification. Decision rules should initially be developed independently of the data and based upon the theoretical perspective of the

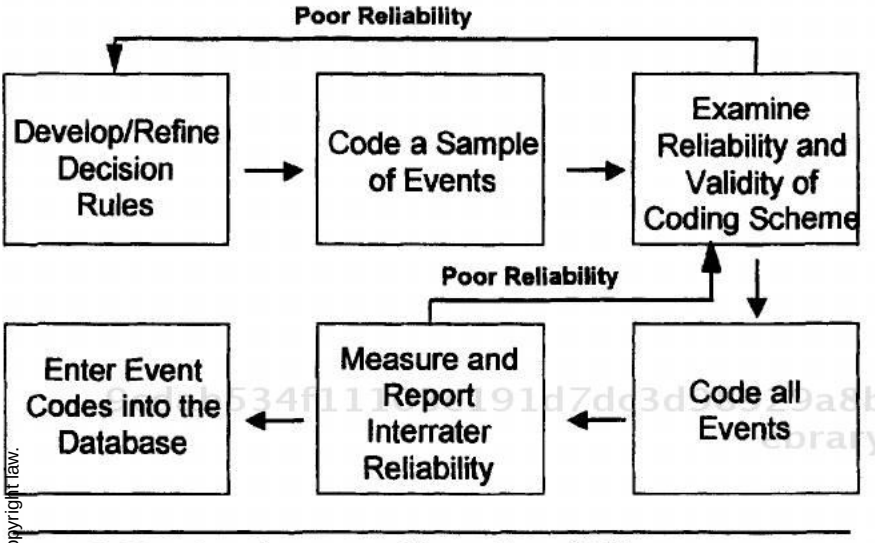

Figure 5.6 Steps for coding events and determining reliability

study. As events are coded, some events may not fit into any decision rule, while others may indicate the need to expand the decision rules to include a new event coding. In the CIP case, we made the development of decision rules an iterative process where we used the rules, read through the set of events, and then discussed whether the coding scheme was adequate and sufficient. If it was not, we added a code or new decision rule and reviewed all events again using the new codes. These discussions about the coding scheme invariably resulted in clearer, more complete, and more valid decision rules.

A few other practical guidelines are useful to make decision rules easy to use. First, be very clear about the boundaries of a decision rule, that is, the incidents that a particular decision rule does *not* include, as well as what it does include. Second, provide examples of incidents that would fall within a particular decision rule along with the rule. Third, keep decision rules short and concise.

Coding a Sample of Events to Measure the Reliability of the Coding Scheme

Once the researcher feels that the decision rules are both theoretically valid and clearly written, the rules should be tested with a sample of events to examine the extent to which they provide a reliable coding guide. One important measure of the reliability of a coding scheme is the extent to which

two (or more) individuals agree on the appropriate codes for a sample of events when applying the decision rules independently. We examine one method for calculating interrater reliability here.

We suggest using a sample of events (at least 25) taken from different chronological points in the database. For example, one might code every nth event or select events at random from different points in the dataset. At least two coders, working independently, should assign codes to each event in the sample using the same set of decision rules. At earlier stages in the development of the decision rules, the researcher can act as one of the coders. At some point, however, an individual who has not been involved in the development of the coding scheme should be asked to code the events. This fresh perspective helps to ensure that interrater agreement is due to the clarity of the decision rules and not just to converging interpretations among the researchers involved in writing the decision rules.

The agreement between the two coders can be measured using a matrix similar to the spreadsheet printout in Figure 5.7. The vector of possible codes should be placed along the top and down the side of the matrix. The codes assigned by Coder A will be placed along the rows and the codes assigned by Coder B will be placed down the columns. Often, an additional category will need to be added to the vector to account for those cases

Coder A \ Coder B	NEG	CMT	EXEC	CXT	NULL	
NEG	5		1			6
CMT		5				5
EXEC			7			7
CXT				6		6
NULL				1		1
	5	5	8	7	0	25

Overall Agreement:	.92
Chance Agreement:	.24
Cohen's Kappa:	.89

Figure 5.7 Coding reliability matrix (using EXCEL)

where no code is assigned. For each event in the sample, the coders compare their codes and add a 1 to the appropriate cell in the matrix.

For example, assume that Coder A in Figure 5.6 assigned an "NEG" code to a particular event and Coder B also assigned an "NEG" code to that event. This implies that a value of 1 should be added to the (NEG, NEG) cell of the matrix which lies on the diagonal. On the next event, Coder A assigned an "NEG" code, but Coder B assigned a "EXEC" code. We would then add a 1 to the (CMT, EXEC) cell of the matrix which lies off the diagonal.

Once the codes assigned to all of the events have been entered, the overall agreement between the coders can be calculated as follows:

$$\text{overall agreement} = \Sigma(r_i, c_i)/N,$$

where (r_i, c_i) represents the value of the cell designated by row i, column i, and N is the total number of codes assigned. In some cases N may be greater than the total number of events, because single events are assigned multiple codes.

While overall agreement provides one measure of reliability, it is also important to employ a statistic such as Cohen's kappa (Cohen, 1960) that corrects for chance agreement. Cohen's kappa is calculated as follows:

$$\kappa = \frac{(\text{overall agreement} - \text{chance agreement})}{(1 - \text{chance agreement})}$$

where,

$$\text{chance agreement} = \Sigma(R_i C_i/N)/N$$

R_i = sum of all codes in row i

C_i = sum of all codes in column i

Values for κ can range from -1 to 1. If $\kappa = 0$, agreement is equal to chance. If $\kappa < 0$, agreement is less than chance and if $\kappa > 0$, agreement is better than chance. Values of $\kappa > .80$ signify reasonable interrater agreement. In some cases, the researcher may decide that certain misclassifications are more serious than others. In such cases, each cell of the agreement matrix may be assigned a weight and the kappa can be calculated to include these weights (see Cohen, 1968 for a description).

We have found that interrater reliability can be low for two typical reasons. First, individual coders may have different understandings of the meanings and applications of the decision rules. If this is the reason for the disagreement, the decision rules should be revised to improve clarity and comprehensiveness and the above process should be repeated. We should

not be surprised if the first few cuts at a coding scheme yield low reliability scores. In our experience, it takes several iterations before clear and unambiguous decision rules that are appropriate for the dataset are developed.

A second typical reason for disagreement is that coders disagree on how a particular event in the database should be interpreted. Since disagreement based on event interpretation does not reflect on the reliability of the decision rules, this source of disagreement should be eliminated whenever possible. One approach to eliminating this source of error is to allow coders to discuss the interpretation of an *event* (not a decision rule) either with one another or with a third person who is familiar with the case while coding. This process can also help the researcher to clarify the wording of events in the database.

Coding All Events and Measuring Interrater Reliabilities

Once the interrater reliabilities calculated for samples of events have attained an acceptable level, the decision rules are ready to be applied to the entire sample of events. This process should involve two or more coders who have a basic understanding of the case and who work independently. The actual coding can be done in at least two different ways. First, both coders can code the entire sample of events and interrater reliabilities can be calculated on the entire sample. Second, one coder can code the entire sample and a second coder can code a *random* sample of events (at least 25) taken from the larger dataset. Interrater agreement can then be calculated on this random sample. Since the second approach does not compromise reliability, it may be a preferable approach for most purposes.

In cases of disagreement at this stage of the research, coders can engage in discussions to see whether or not consensus can be reached on specific events. As was mentioned above, these discussions will often reveal two sources of disagreement: (1) the event was interpreted differently by the two coders or (2) the coders interpreted the decision rules differently. In the former case, the event can be clarified and consensus can usually be reached. In the latter case, the decision rules may need to be revised, in which case the researcher will need to work on obtaining basic reliability with samples of data before coding the entire sample.

Entering Event Codes into the Database

When coding is complete, the event codes should be entered into the keyword column in the database. Event codes should be no more than two or three letters long and codes should be separated by a space or comma.

Step 4. Converting Qualitative Events To Quantitative Bitmaps

Because R:Base© is a database program, its statistical capabilities beyond event counts and frequencies are limited. One important transformation mentioned in chapter 5 was that qualitative codes must be translated into quantitative indicators for statistical analyses. Once events have been coded, the translation from qualitative events to quantitative bitmaps requires only a few mechanical steps to modify the database and convert the data. Note that there are many opportunities throughout this process for small errors. Do not become discouraged by having to repeat certain steps; we have yet to complete this process in its entirety without having to go back to correct some sort of error.

Modifying the Database Structure

After all events have been coded, the database structure must expand to include a unique new column for every code that is selected to classify or represent an event construct (or variable). For example, if 36 total possible codes are applied to events, the table must be expanded by 36 unique columns, each corresponding to one and only one new variable. Because each column is independent of the others, multiple codings for a single event are easily accommodated by this procedure. We suggest that column names be as short and informative as possible regarding their content to eliminate unnecessary confusion. Neither code nor column names can be repeated, nor can they consist of any reserved words (reserved words are protected words that the database program will not allow a user to use).

Updating Code Column Values

New code columns should be defined as integer fields which do not require that any length be specified. The new columns represent the binary indicators of the presence or absence of a particular event construct (e.g., code). Initially, the value of each column for all events should be set to 0 and then updated to reflect the occurrence of a change in a construct for a particular event.[1] Column values are dichotomous and can only be 0s or 1s, representing either the presence or absence of a particular event dimension according to the set of decision rules. As described in chapter 2, multiple

1. Note, however, that although the default value in R:Base looks like a zero (-0-), it is actually a text string variable rather than an integer, and cannot be downloaded into a bitmap. While each program differs, other databases are likely to follow the same default convention.

codes on a single event are created by updating more than one column in an event record with a 1. An example of the syntax entered at the R> prompt needed to execute this step is given below.[2] For this example, the code/column name is *OP*(outcome-positive), the table is named *CIPT* and all events took place after 01/01/76.

```
Update set OP =0 in CIPT where DATE Gt 01/01/76
Update set OP =1 in CIPT where DATE Gt 01/01/76
```

Downloading Column Data into Bitmap Form

Once values are updated to reflect changes in key constructs, the columns of 0s and 1s combine to form the matrix downloaded into a bitmap used by RATS and other data analysis programs. The downloaded data from these procedures should contain each unique event number, the event date, the number of days elapsed since the first event occurred, followed by single columns for each possible codes. The syntax for downloading can be written outside of R:Base in ASCII format and used as a command file. The file should be written in this format:

```
Set Variable start to 01/01/76
Output CIPEXA.DAT
Select number=4 date (date-.start)=4 +
ac=1 ic=1 ir=1 pe=1 tr=1 ci=1 ce=1 op=1 on=1 om=1 on=1 +
a01=1 a02=1 +
From CIPT Sorted By date
```

In this example, actual command syntax is in bold and database variables are in italics. The Set Variable command allows R:Base to calculate the number of days elapsed from the initial event to the current event. This calculated variable is necessary for time series analyses as it establishes both a temporal ordering and scaling of events. The date inserted at the end of the first line (01/01/76) should be the occurrence date of the first event.

The Output command names the output file to be created. Unless a path is explicitly given, R:Base will write the file to its current directory. Using a .DAT extension for the file will help keep different types of files easy to distinguish.

The Select command tells R:Base which columns you want downloaded

2. As with any command in R:Base, the syntax file can be written in ASCII outside of R:Base and submitted to run at the R> screen. To execute a syntax program, type "run [filename]", making sure that the file is either in the current directory or the path is specified.

into the output file, the number of characters allocated to each field, and the order in which they appear. For this example, each event has an ID number, followed by the DATE, the calculated day count, then each of the coded variables. At the end of each line, a "+" indicates that commands continue on the next line.

The From command identifies which table (or file) the data are coming from and the Sorted By command indicates the order of the events in the output file. Sorted By is usually either the event number or another unique field for each event.

Once data are transferred, they must be formated before any analysis can be performed. Because R:Base prints column headings every 20 lines, these headings must be removed throughout the entire file before it can be read into another program for analysis. To do this, either work in a text editor or word processing package to delete the lines, and then resave the file (keeping it in ASCII format). Once the editing has been completed, the bitmap should resemble the sample bitmap shown in Table 5.5 in chapter 5.

Checking the Downloaded Bitmap File for Accuracy

Despite every attempt to prevent errors, it is almost inevitable that errors will appear somewhere in the bitmap. Before any further analysis begins, we suggest that a small sample of events be tested in order to be sure that the column codes (downloaded 0s and 1s) accurately reflect the coding found in a particular incident record. For example, choose 10 or 20 rows from the bitmap. After identifying the unique event represented, examine the codes used to describe its content. Then, check the columns of 0s and 1s following the event number and date in the bitmap. Column values should correspond precisely to coding in the event. So, if an event had three codes assigned, those three code columns should each show 1s rather than 0s. Choosing a small number of events distributed throughout the bitmap and finding them to be correct is a good assurance that the bitmapping process was successful.

Step 5. Aggregating Bitmap Data into Time Periods Using SELAGGR

The bitmap described above contains unique rows for each individual event included in the entire time series. For some types of time series analyses, such as log-linear and logit analysis, a bitmap of unique events is appropriate. If continuous data is required, however, it is necessary to aggregate event counts into temporal intervals as described in chapter 5. This involves

another step of transforming an event bitmap to an aggregated bitmap, reflecting a count of event types occurring during defined time periods (e.g., week, month, year). The format of an aggregated bitmap is similar to an event bitmap but consists of a fixed temporal period (e.g., Year 1, Year 2) rather than an event date or observation number, and columns which represent aggregated variable counts rather than individual observations. The Day Count column in the event bitmap is excluded in an aggregated bitmap. SELAGGR[3] converts event bitmaps into aggregated event bitmaps.

The SELAGGR program is an ASCII text file which contains RATS-compatible commands and requires RATS to operate. However, before an event bitmap can be aggregated, SELAGGR's parameters must be adjusted to fit the event bitmap's content and structure.

Modifying SELAGGR Parameters

The SELAGGR program file must be in the same directory as the event bitmap. RATS© must also be installed on the computer, though it can be located in any directory as long as it is part of the general path. SELAGGR requires parameter changes based on dimensions of the bitmap, so it is necessary to know the exact number of events (rows) in the bitmap and the number and order of columns. These format items are required by SELAGGR and RATS©. Also, columns representing event numbers, dates, and codes must be identified to make sure that event numbers and dates are not aggregated.

Using any text editor, bring up the SELAGGR program copy. Only two sections of the program need to be changed: the beginning "input format" section and the ending "output format" section. No other section needs adjustment. The "input format" and "output format" sections are shown in Figure 5.8.

The file itself is self-explanatory. The OPEN DATA command should be followed by the full name of the event data file bitmap (including its extension and path if not in the current directory). The NOFS variable should be the total number of columns. Our convention for dates has been to count them as three separate columns (MM/DD/YY) and to adjust the format line accordingly (see below). The FIRSTBS variable is the column number of the first variable to be aggregated. Obviously, not every bitmap column

3. SELAGGR was written by Tse-min Lin for aggregating events for the Minnesota Innovation Research Program. Contact the authors for a copy of this program.

```
*****************************************************************************
*                                                                           *
* >>>>> THE FOLLOWING BLOCK OF COMMANDS SPECIFY INPUT FORMAT                *
*                                                                           *
*****************************************************************************
*
OPEN DATA TAPEEXA.DAT  ;* OPEN THE INPUT DATA FILE
IEVAL NOFS=58          ;* NUMBER OF SERIES IN THE INPUT DATA FILE
IEVAL FIRSTBS=6        ;* SERIES # OF THE FIRST BINARY SERIES TO BE AGGREGATED
IEVAL LASTBS=58        ;* SERIES # OF THE LAST BINARY SERIES TO BE AGGREGATED
IEVAL LENGTH=267       ;* LENGTH OF THE INPUT SERIES (NUMBER OF EVENTS)
IEVAL WEEKDAY=2        ;* DAY (0-6) OF WEEK ON WHICH THE FIRST EVENT OCCURRED
*
CAL 1 1 1
ALLOCATE NOFS LENGTH
EQV 1 TO NOFS
 num mm dd yy days $
ac ic ir pc tr ci ce op on om od aid01 aid02 $
a01 a02 a03 a04 a05 a06 a07 a08 a09 a10 a11 a12 a13 $
p01 p02 p03 p04 t1 t2 t3 t4 t5 t6 t7 $
j01 j02 j03 j04 j05 j06 j07 j08 j09 j10 j11 j12 j13 j14 j15 j16
DATA(ORG=OBS,FORMAT='(F4.0,3F3.0,F5.0,1X,53F1.0)') / 1 TO NOFS
*****************************************************************************

*****************************************************************************
* >>>>> THE FOLLOWING BLOCK OF COMMANDS SPECIFIES OUTPUT FORMAT             *
*****************************************************************************
*
DISPLAY(STORE=S1) ### M @-1 ## NE
DO J=1,37
 DISPLAY(STORE=S1) S1 @-1 ## FIX(COUNT(J))
END DO J
DISPLAY(UNIT=COPY) S1
*
DISPLAY(STORE=S2)
DO J=38,NOFBS
 DISPLAY(STORE=S2) S2 @-1 ## FIX(COUNT(J))
END DO J
 DISPLAY(UNIT=COPY) @5 S2
*****************************************************************************
```

Figure 5.8 SELAGGR's input and output sections

should be aggregated. For example, if the event file includes an event number, three columns for the date, and the number of days since the first event, the first variable or series to be aggregated would be the sixth, so "6" would be entered. The LASTBS variable should indicate the last column number to be aggregated. The LENGTH variable should be changed to reflect the number of rows in the bitmap. The WEEKDAY variable can remain 2, unless it is important to the aggregation that day of the week be specified.

Following the input format variables to be changed is a section which

reads the data from the bitmap into the SELAGGR program. The format convention used here is taken from RATS and will be duplicated when creating source code files for RATS. The first three lines of the program (CAL, ALLOCATE, EQV) should not change.

The remainder of this section looks for variable columns to be read. Each variable is reflected with a simple name and must be ordered as it was downloaded from R:Base. A "$" follows each line until all variables have been listed.

The DATA line contains the FORMAT code vital for reading data. The FORMAT section of the DATA line must be only on one line.[4] This string indicates each series' location and its character width. All series must be accounted for in the string. In the example below, F4.0 represents the first series (num) as a 4-digit series with no decimal point. The 3F3.0 represents the entire date field, but is separated into 3 3-digit, no decimal point series (mm,dd,yy). An integer preceding an F code tells SELAGGR to repeat the F coded value. Xs are used to represent empty columns in the file (e.g, 1X represents one blank column). The combination of codes must capture the *exact* layout of the file. Unless this FORMAT code is completely accurate, the program will not aggregate data correctly. In our experience, aggregating errors are frequently eliminated by readjusting the FORMAT string. Be patient, however; detecting errors here can sometimes be frustrating.

At the end of the program, the "output format" section needs to be changed before running SELAGGR. A maximum of 36 series can be aggregated and printed to an output file during any single pass through the program. The number following the "J=1," needs to be equal to (LASTBS - FIRSTBS + 1) in order to be sure that all aggregated data series are included in SELAGGR's output. If more than 36 series need aggregation, the second DISPLAY block must be included (as in the example). However, if less than 36 series are aggregated the "37" should be replaced by the correct number and the second DISPLAY section should be erased from the program. Like the FORMAT line, it is important that this section be completely accurate or data will be printed incorrectly.

Running SELAGGR

Once all parameters fit the event bitmap, SELAGGR can be run using RATS. The current directory must contain the edited SELAGGR copy and

4. The DATA line, however, can be separated by placing a "$" after ORG=OBS, and putting the FORMAT string onto the next line.

the event bitmap. At the DOS prompt, type "RATS SELAGGR.xxx," depending on the exact name under which you saved SELAGGR. Any name can be used for the file. Once executed, SELAGGR prompts the researcher for two items of information needed to complete the aggregation procedure. First, the length of the temporal interval must be specified. The choice of time period depends on the total time span of events, theoretical requirements, and meaningfulness of statistical interpretation. Second, SELAGGR asks for an output file name to be used for the aggregated bitmap. Make sure that the name you choose is different from the event bitmap or the original data will be erased inadvertently.

When SELAGGR writes an outfile with the new bitmap, the first column indicates the period number, the second column indicates the number of aggregated events in that time period, and aggregated variables begin in the third column. This has been a problem for us more than once and is essential for checking data accuracy.

Checking Data for Accuracy

As in all previous steps, it is important to check the aggregated bitmap data to be certain that columns total to the correct sum and are in the correct order. To do this, we recommend choosing a few codes, summing up the total figure in the columns, and checking it against the sum of event types in the R:Base file. If columns do not add up correctly, it is likely that the problem lies in how the data was read into the SELAGGR program or printed to the output file. Check the input and output sections for errors in the FORMAT line, errors in the listed variables and their order, and the number of iterations (in the output section after the J) for printing.

Conclusion

It is our hope that this appendix will make the procedures for coding and data entry more concrete and "doable" for the reader. While the packages mentioned in this chapter have been useful for us, many other similar ones are now available. It should be possible to translate the steps outlined here without too much difficulty.