6304

Using Computers in Qualitative Research

THOMAS J. RICHARDS
LYN RICHARDS

MOST qualitative researchers now work with computers, but relatively few use software designed for qualitative analysis. This is not because they see no need for help in handling rich, complex, or messy data. Rather, computers offer no instant solutions to the problems faced by qualitative researchers, because the data they handle are particularly resistant to tidy processing methods and the methods they use are very unlike the techniques computers easily support. The past decade has produced a plethora of software packages that seem as though they should help, but these are packages designed for executives, librarians, and banks. There is now a much smaller group of programs designed for particular approaches to qualitative research, but they are less accessible and less professionally presented. Thus the researcher is offered a bewildering range of ways of handling textual data on computers, and many of these are quite different from the methods found in qualitative texts. The computer method can have dramatic implications for the research process and outcomes, from unacceptable restrictions on analysis to unexpected opening out of possibilities.

Our purposes in this chapter are to look at methodological features of qualitative data analysis (QDA) to consider how, and how much, and how well, it can be computerized; to give an overview of general-purpose packages that can be used in QDA, and some types of special-purpose QDA packages; to discuss how they can be used and how well they work; to provide some pointers to future software developments; and to stimulate methodological debate on computational QDA. We have written elsewhere of our concerns about the impacts of computing techniques on method and the real dangers of software constraining and distorting research (Richards & Richards, 1991a, 1991b), and of our experiences as researchers making the transition to computers (Richards & Richards, in press). The first remains a background theme in this chapter.

Most reports on software options are accounts of particular programs, usually by their developers and/or marketers. We ourselves are, *inter alia,* developers. (NUD•IST, the software we developed in our research, is now marketed by a company at our university.) Like literary critics who are also novelists, we have a methodological position and a commitment to its products. The reader, thus informed, can evaluate our arguments.

Both as researchers and as software designers, we started from the research processes involved in relating data and theory in qualitative data analysis and the different ways software might support or distort them. This chapter starts there. We then describe and critique a series of types of software in terms of purposes and design, examining the implications of the method supported by each. Thus we offer a methodological map, and, like all maps, it selects the features to be presented.

Our goal is to emphasize the new frontiers, rather than to offer a list of product descriptions.

Product descriptions are readily available from developers (to counter our standpoint): For a special journal issue on this topic, see the November 1991 issue of *Qualitative Sociology*; for conference papers, see Fielding and Lee (1991). Most contributions to those collections (including ours) are arguments for particular software approaches. For penetrating comparative reviews by a nondeveloper and nonmarketer, see Miles and Huberman (1994, Appendix A) and, for more depth, Weitzman and Miles (1994). For earlier partial surveys, compare Tesch (1990) with Pfaffenberger (1988) on sociological approaches, and both with Bailey (1982), Hockey (1980), and Miall (1990) on related software for humanities.

Here instead we offer the researcher a comparative account of software architectures and of the directions of developments. We encourage readers to evaluate software packages in terms of what they propose to do (Did you *want* to do that?) and what new techniques might or might not enhance analysis or restrict method. Having chosen a software approach, the best way to get up-to-date information on software is to send for current product descriptions and demonstration disks and read the survey literature. A list of addresses for the developers of the programs discussed here is included in the appendix to this chapter.

Theory and Data

Working "up" from data is often presented as what qualitative research is especially about. It is done in many ways: building new understandings from "thick descriptions"; reflecting on and exploring data records; discovering patterns and constructing and exploring impressions, summaries, pen portraits. All such efforts have theoretical results. They produce new ideas and new concepts, which are sometimes linked and presented more formally as new theories. Most approaches to qualitative research also work "down" from theory. They incorporate, explore, and build on prior theoretical input, on hunches or ideas or sometimes formal hypotheses. Many also stress the testing of theory derived from the project's data.

Computers easily offer assistance in the management of complex data. They also, with more difficulty, can be used in the discovery and management of unrecognized ideas and concepts, and the construction and exploration of explanatory links between the data and emergent ideas, to make fabrics of argument and understanding around them.

Managing Data

Ideas are produced in qualitative research in heterogeneous ways, many of which are not given the august title of "theorizing." It is not our purpose to survey the range of those methods; rather, we simply note that there is a range of [illegible] methods are supported by software. As other chapters in this volume indicate, different researchers have different methods (and terms) for the exploration and understanding of rich data; production of "thick descriptions" (Geertz, 1973, p. 26); discovery and uses of patterns; construction of new concepts and testing of old; linking of these into theoretical frameworks, explanations, and models; and validating of impressions and conclusions. Nor are these unchanging. Theory testing is emphasized increasingly even in recent writings in the "grounded theory" tradition (Glaser & Strauss, 1967; see also Strauss & Corbin, 1990), which is often, in our view mistakenly, presented as the dominant approach to theorizing in qualitative research.

All these processes involve the recognition of categories *in the data,* generation of ideas about them, and exploration of meanings *in the data.* Because the categories and meanings are found in the text or data records, this process demands data management methods that support insight and discovery, encourage recognition and development of categories, and store them and their links with data. Ease of access to data is important to support recognition of the surprising and unexpected, construction of coherent stories, and exploration of sought patterns, as well as construction and testing of hypotheses (Bogdan & Taylor, 1975). But those methods also must not get in the way, by distorting rich records, diluting "thick descriptions," or demanding routines that destroy insight.

When these theorizing processes were done using manual data-handling methods, researchers often (though by no means always) managed their data by coding for retrieval. The *code-and-retrieve* process consists of labeling passages of the data according to what they are about or other content of interest in them (*coding* or *indexing*), then providing a way of collecting identically labeled passages (*retrieving*). Collecting photocopied segments into labeled hang files and writing text references onto labeled index cards are two obvious noncomputational code-and-retrieve techniques. The technique of annotating passages in page margins is code only.

Before computers, many researchers did not code segments of text. Rather, they felt through, explored, read and reread, "worked and reworked the particulars of ethnographic inquiry" (Kirk & Miller, 1986, p. 32). This required a simpler and more complex form of data management, as researchers compared and systematically built upon data records, keeping growing memo records about the accruing evidence and their exploration of its narrative and convincing body. Theory was arrived at and tested not through the retrieval of text [illegible] dence analysis, including consideration of knowledge about the site or story that is not in the text. For an eloquent account of why code-and-retrieve methods can fail the ethnographer, and of how computing technology rather than the research subject can determine ethnographic method, see Agar (1991)—an autobiography that is the more important because Agar himself is a user of computers and "select-and-sort" techniques.

Many researchers still do not use the code-and-retrieve method. Possibly fewer would use the method now if the software they bought did not support it. But it is certainly the most widely recommended technique for management of rich and complex records. (For different approaches, see, e.g., Hammersley & Atkinson, 1983; Lofland & Lofland, 1984; Lincoln & Guba, 1985; Miles & Huberman, 1984.) However, despite its popularity, the code-and-retrieve method has rarely been examined as a *method*. The literature contains many lucid descriptions of how data records were handled, but reveals little serious debate over what that method of data handling does to data or how it contributes to analysis.

This taken-for-granted method was easily supported by computers and became the basis of most specialist QDA software. Computers, moreover, offered the possibility of addressing its limitations and adapting the code-and-retrieve mode of organization to assist with other theorizing activities. The method was thus subjected to debate in the new context of computing. An odd result of these developments was that the code-and-retrieve method for the first time was treated as in some way atheoretical, merely "descriptive-analytical" (Tesch, 1990). The creation of this dichotomy both underestimates the method and skews critiques of software based on it. One of our arguments below is that all of the specialized software we describe is so based. All of these software packages can be used just for coding and retrieval. And far from merely supporting description, techniques of coding for retrieval strongly support some ways of making ideas, and of constructing and testing theories.

First, the generation of categories, even the simplest descriptors, whether arrived at prior to data reading or by discovery of recurrent topics (Bogdan & Taylor, 1975) or *in vivo* categories in text (Strauss, 1987), is a contribution to theory. Decisions are being made about what is a category of significance to the study, what questions are being asked, what concepts developed, what ideas explored, and whether these categories should be altered, redefined, or deleted during analysis. Second, decisions about what text segments are relevant to a category are never merely clerical decisions; they always involve some theoretical consideration. Third, the viewing of segments from many documents on one topic or selected topics always offers a new way of seeing data. This is the [illegible] claim of the method to support analysis, and researchers using it clearly engage in the building up of theories. Moreover, the method supports pursuit of patterns by comparison of text segments on that topic from different sources (e.g., Did the young women have different ideas about domestic duties from women of other age groups?). Such questions may be crucial for locating patterns and are sometimes formally portrayed by presentation of data in qualitative matrices (Miles & Huberman, 1984).

So it is misleading to label the code-and-retrieve method as not theory building. But the challenge remains to adapt it to ways of recording, linking, exploring, testing, and building cumulatively on the insights derived from data. To draw on a distinction first made by Turner (1981), theory *emergence* in qualitative research is interlinked with processes of theory *construction*. Ideas, concepts, and categories discovered in the data are woven by researchers into fabrics of theory. These processes offer greater challenges to software designers.

Theory Construction

Theory *construction* in qualitative research (the exploration and linking of theoretical and other organizing and explanatory concepts and statements) is creative, not merely mechanical. The data-handling tasks associated are thus highly complex. And theory *testing* is usually part of theory construction, not a subsequent stage. Concepts are captured; links are explored, created, and tested; ideas are documented and systematically reworked, in textual memos, models, and diagrams expressing the specification, explication, exploration, and elaboration of theories. How can computers support this?

The code-and-retrieve method, we have argued, supports theory emergence. It also expresses theories that can be represented by codes and then tested by looking for codes in text and studying the relationships of codes. Computer-based code-and-retrieve will do this better, because computers are good at working with structure, not content. In a code-and-retrieve system, we express or define content by coding the text.

Suppose, for example, you have a hypothesis that people of a certain type and in a certain situation will behave in a certain way, and you have comprehensive interviews with individuals that check in each case for the types and conditions and the behavior, and the interview transcripts are coded for these. An example is "Young mothers who are reluctant to return to work explain it in terms of a woman's duty to stay at home." Then a simple study of coding patterns in the interview texts can be used to confirm the theory's *correctness* (check if there are any interviews coded for all the types and conditions but not for the behavior) and *completeness*—"*Only* young mothers . . . "—(check if there are any interviews coded for the behavior but not for all the types and conditions).[1] Note that what we are doing here is successfully using co-occurrences of codes within interview documents as evidence for features of theories: *Textual structure* as delineated by code-and-retrieve methods can be related to *theoretical content* (including information about the world). If this were not so, we would not use code-and-retrieve methods.

But this is not always so. Most social science theories find their support in the *content of the data,* not the *structure of textual records.* Management of records by use of code-and-retrieve in such cases offers help, but that help is limited to retrieving all passages coded with something relevant to the theory in question, so that the researcher can reflect on them all together.

This is not an insignificant contribution. The ability to retrieve all the text about a certain topic or topics strongly supports the development of new insights. The computer can do this quickly and efficiently. Sophisticated programs offer a wide range of ways of selecting retrievals according to co-occurrence or non-co-occurrence of codes in text, allowing the researcher to "fracture" the data (Strauss, 1987) and see it anew. But this contribution to the researcher's ability to access data should not obscure an important distinction: that between the *textual level* of work, which is where code-and-retrieve methods operate, where we code for talk-about-return-to-work and talk-about-mothering, and the *conceptual level* of work, where theories about people and the world are expressed, where evidence and argument are brought to bear, and where returning to work and mothering are explored.

The code-and-retrieve method we have described applies only to text. That which is coded and retrieved is the document. Literally, one codes talk-about-mothering (the text passages), not mothering (the concept). But no researcher stops there. What one would *also* like to do with software is to support directly conceptual-level work, not just textual-level work—that is, to have software that could directly represent the concept of equality and how it gets related to the concept of parenting.

So the dichotomy that matters is not descriptive-analytic versus theoretical: All data management methods involve theorizing. Rather, in assessing what computers contribute we need to distinguish textual-level operations from conceptual-level operations. Whereas code-and-retrieve as we have described it is a textual-level operation, one's codings and retrievals are guided by [illegible] theory, and (inevitably) put theoretical blinkers on one's access to the text. Textual-level operations are theoretically relevant, but they do not construct or operate on theories.

Finding ways of supporting theoretical-level operations in qualitative research offers a major challenge to software designers. Consider, for example, how we work when developing theory from the text. We often get going by finding little things that relate in some meaningful way—perhaps, if our interest is in stress, that certain topics get discussed in anxious ways (and that is something that good coding and retrieval can find for us). So then we start looking for components in those topics that might cause anxiety, often by studying the text, finding or guessing the components and coding for them, recalling situational facts not in the text, and looking for suggestive co-occurrences of codes. We might on a hunch start looking at text passages on people's personal security and how they arrange it (research on background theory here, and lots of coding again), to see if there is some possible connection between components occurring in the anxiety topics and security arrangements. If we find one, the theory is still thin, so we embark on a search for others, and thereby look for a pattern. The result of this is a little group of chunked-together coded text, ideas and hypotheses that, provided they can be kept and accessed as a chunk, can become an ingredient in further more abstracted or wide-ranging explorations. This chunk is said to be of larger "grain size" than its component codings, and it may in turn become an ingredient of a later theorizing of larger grain size still that is built out of existing chunks. (Big fleas are made out of smaller fleas.)

And so the web—of code, explore, relate, study the text—grows, resulting in little explorations, little tests, little ideas hardly worth calling theories but that need to be hung onto as wholes, to be further data for further study. Together they link together with other theories and make the story, the understanding of the text. The strength of this growing interpretation lies to a considerable extent in the fine grain size and tight interknittedness of all these steps; and the job of qualitative data handling (and software) is to help in the development of such growing interpretations.

This network of concepts, evidence, relations of concepts, coordinations of data, of hierarchies of grain size where the theory/data/explanation chunks of one grain size are the data for the work of the next grain size up, is a good fractal-like model of people's explanatory belief systems (belief systems are explanation systems). This is how a person (e.g., a social scientist) reflectively constructs an explanation, a story, for and from data.

The process is not all bottom-up, however. The researcher uses at each stage expectations, prior theories, hunches, experience, and a good education (as with the theoretical determination of textual codes). The network builds up from the bottom, guided by a vision of the structure of a larger-scale network into which these small empirical gleanings must fit. When one gets there, the larger-scale structure is likely to be different in many ways from the early ghostly vision; were it not so, the constructed theory would be quite unempirical, quite unconditioned by one's data. And if one's prior ideas are wildly out, then that will show up in the increasingly procrustean strains of trying to build the anticipated larger structures from the small, heavily data-conditioned ones. Here is where one's critics will show one a more amenable approach to interpreting the data, fewer exceptions requiring fewer ad hoc justifications, more meaningful relationships binding cases and patterns together, more elegance.

We will call this description of building relations between data and theory *data-theory bootstrapping*. Providing direct conceptual-level support for this process puts some interesting demands on software design. Coding for retrieval seems to be basic to such procedures, but researchers also want to hold their growing nets or hierarchies of concepts, evidence links, groupings of ideas, and so on that make up the explanatory structure in an accessible way that will help them see where they have been and give access to the fine grains out of which we build the larger grains. The software system that would help with this would hold not just the data and tools for manipulating it, but also in some sense the growing analysis and explanation system.

And because, in that recursive fractal-like way we have described, the partial results and little theories become part of the data for the next move in the analysis, that software would treat the analysis/explanation material added to the database as more data alongside the original textual material. The very analytic structures, the explanations, become more data. Indeed, the very processes of analysis (the computations) should be fed back in as data. That is, we want to save as data the theory/data/explanation chunks of one grain size so they can be explored as data for explanations at the next level up. Methodologically, this is known as *system closure*: Results obtained about the system, analytic techniques used on the system, become part of the system. A hallmark of qualitative social science research (but not of physics) is that the data being researched in a project are closed over its own techniques and results. System closure is the software feature needed to support directly the conceptual process of data-theory bootstrapping. (Needed, yes, but not sufficient—system closure will not necessarily give you a leg up on direct conceptual-level software operations.)

Qualitative researchers also need to jump from one code to a *conceptually* related one (to explore theory) or to a *factually* related one (to explore patterns in the world the research is about). So the database should maintain and exploit theoretical links between concepts, and the real-world facts about and links among people, places, actions, and so on, not just explore *textual* links between codes representing those concepts, people, and so on. If, for example, we store that John is married to Mary, that John is a blue-collar worker, and that John has been out of work (note that these are facts that might not be expressed in the text at all), we should then be able to ask for all the remarks of women married to out-of-work blue-collar people on some coded topic and *automatically* get Mary's remarks.

The procedures of theory construction described here require above all a very flexible, very easy-to-modify database, that will shift, reorganize, undo, and backtrack to earlier states. This is because the process of constructing an understanding is tentative, involving the exploration and testing of hunches at all grain size levels, hanging onto them if they look good for now, throwing them away when they no longer fit, while maintaining the rest of the growing structure.

Can computers assist with, even improve on, the ways we construct and test theory? Can they go further, and support the *explicit formulation of theory*? What of the explicit *finding and recording of knowledge* about the situation being studied, the putting of data to theory?

Current Situation in Qualitative Software

In the following sections we explore the architecture and purposes of available software. We start with types aimed at a broad class of users, but that can offer advantages not available in specialist packages. This provides a basis for understanding why special-purpose packages arose and what they try to do. We then deal with those, considering first software that adapts the traditional code-and-retrieve method, and then approaches that combine that method with new ways of constructing the links among theories, knowledge, and text, testing them and modeling them.

General-Purpose Software Packages

Word Processors

Apart from the obvious and familiar advantages (such as ability to inspect an entire document, collate and explore selected extracts in a new document, print it out, line number it, edit it), the modern word processor (WP) offers some features unmatched in most specialist QDA software. If the data are textual (e.g., interviews) and in WP document form, these features include the following:

- The ability to handle multiple documents on-screen in separate windows at the same time, which facilitates comparing thematically similar passages in different documents and copying segments of one document to another.
- The ability to handle formatted files (using the WP program's own format conventions) so that tables, diagrams, and the like can be included. Most special-purpose QDA packages work only with text-only ASCII files.
- The ability to include static pictures, charts, tables, and so on as illustrations or as editable models of the emerging ideas and diagrams of the theories. These need not be computer generated, but can be documents of any sort read into a disk file by use of a scanner, and may be in color.
- The ability to include video and audio data, accessible via icons in the WP text.
- Generally good text-search facilities, which in some WP programs support the use of patterns in text search.
- A *publish-and-subscribe* facility, in which a passage from one document is marked as available for inclusion in others (published). When included in another (subscribed to) it is not copied; rather, it is as if the published portion of the first document is visible directly in the second document. In this way, if the published passage is edited at any future time in the first document, those changes show up in the subscribing document.
- A *linking* or *hypertext* facility by which the user can select the subscribed passage and so open a new window into the publishing document at the published text, for inspection and editing. An elegant application of linking, for QDA purposes, is to mark passages in a (publishing) document with keywords or icons, and link the keywords through to the subscribing documents. Selecting the keywords in the subscribing documents will then open a window into the publishing document at the position of the keyword, so the user can see the keyed passage. In this way groups of related passages, in the same or different documents, can get linked together and the user can jump from one to the other.
- An *annotation* facility, in which an icon is inserted in the text and clicking on it opens a text window in which one can read and write memos. The annotations can optionally be printed with the WP file at the point they annotate.

These relatively recent features, such as publish/subscribe, linking, incorporation of video/audio data, and annotations, powerfully extend the more traditional WP features such as text search. For the qualitative analyst they provide imaginative ways of linking data, combining different media appropriately, and relating commentary and theoretical memos.

The main problem with WPs is what they do not do, or support badly. They do not automate the grouping of similarly coded passages—one must copy and paste, or link, them oneself (or at best write a macro). They become very clumsy if one tries to use them to handle large numbers of codes or many references from codes to text. They will not provide text searches for co-occurrences of codes (more on these in later sections). And they will not provide clerical and management tools (e.g., What codes have I used? What do they mean?). In WP programs, clerical data must be stored separately to prevent it getting lost in the data documents, whereas good special-purpose QDA software will hold and retrieve clerical data where and when wanted, to facilitate database exploration.

Nevertheless, smaller projects in particular may welcome the modern word processor as a flexible and full-featured tool for document exploration and the construction of analysis documents that relate themselves neatly to source documents and other media, which can be only a mouse click away. And specialist software packages too often lag way behind in these features.

Text Search Packages

Text search, long ignored by qualitative researchers, offers much more than useful tools for linguistic or protocol analysis. It can find themes in the text, gain instant access to occurrences of a newly discovered theme in text already coded, and locate topic markers such as question numbers.

The principles of most text search tools are much the same, and extend the text search facilities found in WPs. The simplest will search for a string (sequence of characters) in files of text and report each find in some way, such as [illegible] it in context and outputting its character or line position in the file. A more sophisticated type will support search for Boolean co-occurrences of strings (*and, or,* and *not* searches) within some stated unit of text, such as a line or paragraph, or even allow one to express complex *patterns* in the search, and will report and save every string in the text that matches the pattern. The common grammar used to express patterns is called *regular expression syntax* and is embodied in the famous Unix utility called grep (global regular expression printer), which is usually available as freeware for other types of computers, and is more powerful than most proprietary text search packages.

Text search packages search files (they do not need to be open in a window) and sometimes have special facilities for fast searching for user-supplied keywords in documents and then providing statistically useful results of various sorts on keyword co-occurrences and correlations.

Text search alone is not a sufficient tool for most qualitative researchers, because they also want, among other things, to store finds at a code (a system closure feature). But it is a necessary tool for gaining direct access to data records, rather than accessing them only through codes expressing the researcher's interpretation. When words of the text matter, or codes fail, text search is essential, and the computer searches text much faster than one can code it. Hence software supporting other QDA methods is greatly strengthened if it includes text search facilities.

In addition to general text search packages such as GOfer™ and ZyINDEX™, a number of concordance and similar content-analysis programs have been developed, primarily for literary studies, that will carry out word frequency counts, provide listings of chosen words embedded in a line of context and with a references to where they occur (KWIC—keyword in context—indexes), spot grammatical styles, and usually provide statistics on their finds. Some of these packages are extremely sophisticated (for further details, see Bailey, 1982; Hockey, 1980; Miall, 1990).

Relational Database Management Systems

Relational database management systems (RDBMSs) can undoubtedly be very useful in a social science project, for both management of project information and analysis of research data. However, their powers are often misunderstood and misapplied in the QDA context.

Suppose you have a card file of your interviewees, each card containing name, address, gender, birth date, and date of interview. These cards can be easily replaced in an RDBMS by a two-dimen- [illegible] for name, address, and so on. The rows are called *records,* corresponding to each card in the original stack. The columns are called *fields.* You define the fields for each table, then create as many records with those fields as you need. Typically, fields can be defined as *numeric* (holding a number), *Boolean* (holding *true* or *false*), *character* (holding a few words of text, such as a name), *date,* and *memos* (holding your notes on the record). You have to specify how many characters each character field occupies (except memos, which is usually set to some upper limit, such as 800 characters). Whether or not a field is filled in for a given record, it will still occupy that number of characters in disk storage (except for memos, which can grow up to the limit).

The power of database systems comes from tools to *sort* records on any numeric, Boolean, or text field, or combinations thereof, and to *filter* records, extracting certain ones with desired values in various fields. If you think of other fields you want after you have created your database, it is usually easy to add them in. Some RDBMSs also specialize in handling text rather than fixed-size numeric or character fields; these can be of advantage for QDA purposes. Facilities are often provided for text search on text or character fields, but note that text sectors of these fields cannot be coded for retrieval of the coded segments.

In your interview project, having created the database table described above to manage biographical data about interviewees, you could then create further ones to handle data about what they said in the interviews. If the interview comprises a number of questions, with free text answers to each question, a common procedure is to create a database table for each question, one record per interviewee, with a field for the interviewee's name, another containing the entire text of his or her response, and further ones labeled with topic codes containing the portions of the response germane to each topic code.

These database systems are called *relational* because the researcher can relate one such table to another. All he or she needs is to have a field in common. Any of the tables above can be related if they all have the interviewee name field in common. Similarly, tables with a topic field in common can be related through the common topic field, allowing the easy extraction of what an interviewee, or selected interviewees, said on that topic in answer to different questions. The result is that the researcher can use tables jointly to extract interesting data. One could, for example, list all married female interviewees who have a certain attitude toward alcoholism. This enables numerical and comparative studies—What fraction of all married female interviewees are they? Are unmarried females [illegible] different [illegible] attitudes?

So how useful are these systems to a qualitative researcher? They work best for discrete structured data, rather than for long, unstructured textual data requiring close study of content and data-theory bootstrapping. The attempt to create fields corresponding to topic codes, and putting text in those fields, is extremely expensive of storage. Moreover, if one uses many codes, more code fields per record tend to be empty, leading to sparsely filled tables that are hard to work with. RDBMSs work well for such purposes as analyzing the results of structured questionnaires that get discrete data as answers—names, places, and so on—or for analyzing social systems that can be described in discrete terms (participants, objects, transactions between, and so on). After all, RDBMSs have grown up to handle the discrete data of businesses—employee data, inventory, sales transactions, and the like—for purposes of analysis of business trends.

Like many general-purpose tools, however, they can be ingeniously extended. One such extension is a powerful technique for the construction of comprehensive relational databases known as the entity-relationship approach (ERA) (Chen, 1976). This approach comes into its own when the subject of the research project can be characterized as a system whose operation is to be studied, such as a classroom situation, a workplace, or a household. The user draws up a network diagram in which the *nodes* (the "knots" in the network that the lines join) are the various entities under study, such as the personnel and departments and functions in the company, the means of communication used in the company. Any relations among these entities are drawn as labeled lines (*arcs*) linking the nodes, for example, "reports to," "communicates with," "uses."

In this way the network diagram will specify and relate the major activities and entities in the system, such as the people involved, their tools, their goals within the system, their choices among tools, their actions. If the qualitative data about each of these features (nodes and arcs in the net) tend to be discrete rather than narrative (e.g., for an activity: type of activity, date and time of its occurrence, tools used, participants, its goals), then a whole database table can be set up for that node, whose records hold data about each item of that type that is observed in the study. Observation of the system (studying classroom activities, observing the shuffling of information around the office, and so on) then provides the data that go into the records.

The links in the network diagram show how to relate the database tables in the linked nodes to each other (the *relational* aspect of an RDBMS), and then the very powerful browsing features of a good RDBMS package can be applied to study [illegible] the operation of the system under study. Winer and Carrière (1991) provide a very instructive and lucid account of a highly innovative system using RDBMSs in this way.

Where data are often discrete and the subject of study can be thought of as a system, the ERA diagrams provide a powerful discipline for creating semantically clear and precise network diagrams describing the system. No meaningless arrows or confused categories here. Then, using the ERA to create a relational database for the data provides a powerful data analysis system for the researcher. But a word of warning: You will want on your research team a computer scientist trained in *data modeling* (construction of ERA diagrams that can be turned neatly into an RDBMS system); the task of system analysis necessary to set up the database system is a skilled professional process. A better idea might be to start teaching data modeling in sociological methods courses—that might help to critique the current often meaningless use of diagrams in sociological literature, and to develop powerful skills in representing social systems and modeling theories.

HyperCard® and Hypermedia

The popular Apple® product HyperCard is a nonrelational database management system with an appealing user interface. A table of records is represented as a "stack" of file cards, only the top one of which (i.e., one record) is visible at a time. One can easily design the visual appearance of card stacks using HyperCard's simple drawing and design facilities. Typically, this is done by designing "fields" to hold the desired data on each card. "Buttons" can be added to the cards that, when selected by mouse click, carry out some predefined actions, such as displaying the next card in the stack. A simple and rather weak programming language allows "stackware authors" (don't call them programmers) to program the behavior of cards, especially button actions.

This simple software metaphor lends itself to some clever applications for QDA. You can tell the products by their Hyper-names, but don't assume they all do the same thing. Hyperqual is a simple code-and-retrieve program (Tesch, 1990). A sophisticated Scottish newcomer, HyperSoft, ingeniously addresses modeling tasks. HyperRESEARCH, discussed below, takes a specific approach to hypothesis testing. These all have in common the restriction of displaying only one record at a time, and none can act as a *relational* database, because there is no simple way in HyperCard of relationally linking stacks by common data fields.

can add scrolling text fields to cards. These fields can support text code-and-retrieve facilities, with the advantage of "one-step" coding (no need to code paper records and then input coding data). Moreover, by positioning buttons over words or phrases and programming the button action to go to other cards with the same words or phrases, one can provide a sort of *hypertext* facility—a way of linking similar text passages so that one can move from one to the other. (Where the links are not just with text, but, for example, with audio and video media, we call this facility *hypermedia*.) The hypermedia facilities of HyperCard can be exploited to link field data text to the researcher's memos about it, or to records of associated factual data, or to link the passages of the research report to evidence material relevant to each passage. Other packages, such as StorySpace™, are designed to support these facilities directly, and should be taken seriously as tools for imaginative exploration (and creation) of text. For an example of such work using another such package, NoteCards™, see Halasz, Moran, and Trigg (1987). For a general survey of hypertext principles and software, see Conklin (1987).

Conclusion

Software not designed for QDA can be useful for certain purposes, but it can also constrain researchers who need flexibility and multiple methods. However, a study of how these systems work is essential if researchers are to know what they might expect of specialist software. From these general-purpose approaches, specialist qualitative research software should now gain such features as the following:

- publish-and-subscribe facility as a way of maintaining segmentation (coding) of text that changes; for example, as one adds commentary directly into the field notes
- pattern-based text search, *plus* the ability to code the finds automatically
- the way RDBMS packages organize discrete qualitative data; sort, filter, and make reports on it; and can be used to find patterns in the data
- hypermedia features that support "commenting" on segmented text and other media data directly, associating database material and audio/video playback with text, and storing memos linked to text, then moving easily among memos, data, and text

Special-Purpose Software for QDA

The 1980s delivered a collection of QDA software tools designed to address the peculiar needs of QDA work. Recent software systems build on the techniques developed by the pioneer programs, and incorporate both their ability to do the tasks of coding for retrieval and, we would argue, the disadvantages of that method. We distinguish five types of specialist software, each identified by its information representation and processing methods. The first, code-and-retrieve, is a form of information processing that is incorporated in each of the other four. All of the later types provide other ways of storing and accessing knowledge and constructing, exploring, and testing data and theory. In each case we describe one (sometimes the only) software example.

Code-and-Retrieve Software

This type of software was the first development for QDA, created by social scientists attempting to replicate the code-and-retrieve techniques that they had used manually. It has been around long enough for studies by or of its users to appear (e.g., Tallerico, 1991), and the range and operation of packages then available has been fully described by Tesch (1990).

Code-and-retrieve packages, all in different ways, allow one to enter (and change) coding of specified text segments of documents into a database, then collect and display all text segments marked by the same code. Some have enhancements that improve considerably on manual methods. The first available and best-known example of this type of software is the Ethnograph (Seidel & Clark, 1984). In its forthcoming version 4, the Ethnograph will do the following:

- retrieve on presence or absence of two (or more) codes; that is, report and optionally display all text portions indexed by all of the nominated codes, or by one but not the second—doing so-called Boolean searches using logical *and* and relative *not* or searches for sequences or proximity of codes
- support the collection of documents into sets, called *catalogues*; retrieval operations can then be restricted to a chosen catalogue
- do text search
- store memos

- display the occurrences of codes in files or specified text portions
- display subheaders to identify speakers or context
- display statistics about the number of retrievals
- hold factual information about each document or codes applying to the whole document (called *face-sheet variables*) or to individuals (in a *speaker sheet,* recording religion, gender, age group, and so on) (These codes can be used in multicode retrieval. Note that a face-sheet variable, though conveniently indexing a whole document, is actually coding a *fact,* and so operates at the conceptual level as well as the textual level. Imaginatively used in retrievals, this provides a powerful way of relating conceptual-level operations to textual-level ones.)

The method thus offers much assistance in managing data, and also, as we argued above, in building and using theoretical categories. But it also has major problems, and software developments have sought to address these. First, the method "decontextualizes" (Seidel's term, used in Tesch, 1990, p. 115). Stripping the segment out of context is necessary and desirable if it is to be "recontextualized" in the new category context. But the context of data is essential to any "holistic" interpretation. Second, the method always threatens with rigidity. All code-and-retrieve software permits introduction and deletion of codes at any time, but this leaves problems in constructing new categories after the coding of many records without those categories; for example, how to return to the passages previously missed? And third, this method tends to impose on qualitative research a chronology more like that of survey research: sequential stages of data collection, data coding, then data analysis. Analysis is postponed if researchers find difficulty in keeping the ideas and insights emerging while clerically coding (and computers will do much more coding, so the task can become more dominating).

Developers of software supporting code-and-retrieve usually and rightly deplore these effects, particularly the last, and attribute them, particularly the last, to bad habits in the user rather than the software. Each of these problems is accessible to computer solution, but, like most manual systems, software systems do not easily support the integration of the process of coding (often perceived as dreary and clerical) with the (tentative, exciting) processes of discovery and surprise, or recording of new ideas and exploration of links between emerging categories. Developers make no claim that code-and-retrieve software supports anything like the entire qualitative research process, but minimally that it speeds up and extends the common clerical business of document coding, and makes the clerical business of retrieval guaranteed complete relative to the coding (unlike flicking through pages of transcript looking for marginal annotations). It is probably this perception of code-and-retrieve software that has led to the mistaken view that it has no theory-finding, theory-building, or theory-testing ramifications.

Software is certainly responding to these challenges. "Decontextualized" text collected at a code was always easy to chase back to the original context via information about the location of segments; but software using multiple window interfaces will allow the original documents to be viewed alongside the grouped retrievals. Limits on the number of codes available and/or the number of times a given rich passage can be coded are being extended or even removed. Considerable effort has gone into making the codes and their contents flexible, so data segments can be easily recoded and inconsistencies in coding discovered, and codes viewed, redefined, amalgamated, deleted, and duplicated *safely.* Retrieval styles are now more flexible and include exploration of context by sequencing or proximity of codes in the text. Questioning is no longer limited to the *intersection* of codes at particular text segments. (*In document co-occurrence* is often more important, to find documents coded somewhere with specified codes, though the segments coded do not intersect.) Storing *knowledge* about the situations or people or behaviors studied is often supported, even if that knowledge does not refer to whole documents. And recent software assists researchers, as filing cabinets never did, in managing codes and in the storage of ideas *about* codes and data, in memos, related both to the codes and to the data, as well as in checking reliability and consistency of coding and coders. Some systems combining code-and-retrieve with text search allow automatic indexing of text finds, and a few offer pattern-based text search, essential if the text does not conveniently always offer exactly the characters sought.

Coding for retrieval is one procedure incorporated, increasingly with extra facilities, in virtually all sophisticated QDA software, because it is one very major type of software support that most forms of QDA need, and that general-purpose software cannot provide easily if at all. But the method retains the limitation we stressed in the early sections, that the code-and-retrieve method directs analysis to occurrence or not of specified codes at selected portions of text.

Rule-Based Theory-Building Systems

One direction has been to seek ways of more explicitly specifying, developing, and, especially, testing theory. In commercial expert system software, this is often done using the idea of a *pro-* [illegible] An example of this genre is HyperRESEARCH, a tool that shows what HyperCard can do if you really work at it (Hesse-Biber, Dupuis, & Kinder, 1991). In its fundamentals this is a code-and-retrieve system, but it exploits the Macintosh™ computer and HyperCard to include pictures and audio- and videotapes among the documents it can index. It also contains many of the desirable enhancements to code-and-retrieve technology we nominated above. It will do text search with "autocoding" of the finds. It will do Boolean searches for in-document co-occurrences of codes, not just for places in the documents where codes intersect. But, significantly, it allows one to retain retrievals in the system. Like autocoding of text search finds, this is a significant system closure feature.

It does this through the use of *production rules.* A production rule is an if-then rule of the form "If conditions C_1 to C_n hold for some data, then perform action A on the data." In HyperRESEARCH, the form is "If a case is coded as C_1 . . . and C_n, then code it also as A." (HyperRESEARCH looks at its data in terms of *cases,* rather than documents, and tries to find theories that explain all the cases studied. Textual data for the cases could be split across different documents.) The new code A (called the *goal*) is then added to the database, referencing the cases coded with *all* of C_1 to C_n. Once the conclusion code A is in the database, it can be used as a condition for another rule, which the user can then begin constructing.

Alternatively, one can build up rule sets "backward," beginning with a rule expressing one's overall hypothesis, then trying to find rules whose conclusions are the conditions of the hypothesis. This process is repeated until one arrives at rules whose conditions are all codes in the textual database. Running the rules forward then enables one to find cases where, by virtue of having all the right initial conditions in their text, the conclusion of the ultimate (initial) hypothesis also holds, even though it is not coded in the text. In the example given by the developers (Hesse-Biber et al., 1991), the overall hypothesis is that if a mother has a negative influence on her daughter's self-image (C_1) and the daughter dislikes her appearance (C_2), then the mother has damaged the daughter's self-image (A, the goal of the research). Then if this C_1 and C_2 are not codes used in the research, they may be defined as goals of further rules, such as, if the mother is critical of the daughter's body image and the mother-daughter relationship is strained, and the daughter is experiencing weight loss, then add that the mother has negatively influenced the daughter's self-image (C_1). In this way one creates "chains" of rules backward from the goal until one is using only conditions that are already codes in the case documents. At that stage the rule set can be "run" to find how many of the cases end up with the goal [illegible]

This sounds rather like a knowledge-based expert system in artificial intelligence, in that it contains qualitative production rules. But taking the rules together, it amounts to a search for the cases that have coded *somewhere in them* all the conditions of all the rules that are actual indexing codes (i.e., in-case co-occurrences). If (continuing the example of the previous paragraph) the initial codes in the documents used in the rule set are K_1 to K_m, we are doing an in-case co-occurrence search for these codes, nothing more. Cases where the search succeeds are treated as confirming the final hypothesis "If C_1 and C_2, then A," and those where it fails as (presumably) disconfirming it.

But be careful of the methodology here! The *disconfirming* instances of a hypothesis "If C_1 to C_n hold, then A holds" are cases where *all* the conditions hold and the goal, A, *does not* hold. Cases where not all the conditions hold are not disconfirming instances at all. Typically, C_1 to C_n are a mixture of theoretical statements and specific conditions that hold for a given case, and A is an *observable* feature of the case. In other words, *it must be possible* to evaluate whether the goal holds in a case *independent* of whether the conditions all hold. But given that there is no independent coding of cases for A (rather, A is added whenever C_1 to C_n hold), we can never find a case in which C_1 to C_n hold but A does not—the hypothesis is a tautology. What we *can* find is cases where not all of C_1 to C_n hold, but that proves nothing about the hypothesis. In fact, what the running of the whole rule set boils down to is simply looking for cases where the original codes K_1 to K_m occur!

What, then, is the value of production-rule systems for QDA? These rules are certainly an intuitive way of articulating at least some sorts of theory. They do provide a way of bridging the gap between textual-level analysis and the representation and analysis of facts and theories to which we have drawn attention. Starting at the textual level, the production rules allow the definition of increasingly abstract and theoretical concepts in terms initially of the textual codings (K_1 to K_m in the above example) and ultimately whole theories as the later production rules. This conforms closely to the model of data-theory bootstrapping, and provides an elegant way of relating textual-level and conceptual-level operations. But for the process to be of any value in theory testing, as distinct from theory construction, the cases must *independently* be coded with all the rule goals, and the rules run as a search procedure for cases where all conditions of a rule hold but the goal does not.

A methodological difficulty with this is that production rules are supposed to bridge the textual/conceptual divide by making their goals (A's) be more theoretical concepts defined in terms of less theoretical existing ones (the C's). Now we are saying the A's must be observable features already coded into the text.

Note carefully that this argument is not so much critical of HyperRESEARCH, which provides powerful additions to code-and-retrieve and the incorporation of production rules, as it is of thoughtless ways of employing the production-rule facility. In qualitative research, such misuses could contradict the central goals of building up understanding from data by forever returning to it. This is not easily achieved by getting a machine to insert new codes when it finds others, without care to see if the insertion of the new code is justified by the text. There are also dangers of building, in any software, an edifice of sophisticated reasoning on textual-structure coding. A weak link will always be the adequacy of the coding process, and this caution applies to all the following sections.

Logic-Based Systems

Discussion of production-rule systems leads naturally to logic-based systems. These use if-then rules for their representation of hypotheses, as the production-rule systems do, but the type of rule and the way it works is very different and more sophisticated. The rules are those of *clausal form logic,* a computationally useful way of expressing and computing with the standard calculus of formal logic (Richards, 1989). A useful fragment of clausal form logic lends itself well to computer implementation, both to represent data in a way that is an alternative to RDBMSs and to compute with those data using logical deduction. This computational paradigm is known as *logic programming* and is realized in the computer language Prolog (Clocksin & Mellish, 1984). The best-known examples of its employment for QDA purposes are in AQUAD for IBM-PC computers (Huber & Garcia, 1991) and QUALOG for mainframes (Shelly & Sibert, 1985), on which AQUAD was based. AQUAD is not only written in Prolog, but makes Prolog available to users to express hypotheses and compute with them. QUALOG uses a different logic programming language, LogLISP. We will discuss only AQUAD here as our exemplar of this genre.

Like nearly all QDA systems, AQUAD supports code-and-retrieve. However, it provides a sophisticated set of retrieval patterns, called hypothesis structures, used in *linkage analysis.* Although some of these retrieval patterns are Boolean, such as looking for one code or another in a text, many are more interesting, such as searching for positive *and negative* cases of one code occurring *within a certain distance* of another in the text. The output of such searches is typically numerical tables showing cases where the searched-for linkage did hold and, for instance, the textual distance between the codes. This is why the linkage analysis [illegible] and not just bare retrievals of text. The flavor of linkage analysis is not "Show me all text of codes A and B within textual distance d," but "To what extent do codes A and B occur within textual distance d—is it a significant association?" This is a very powerful feature that helps link qualitative and quantitative analysis in one research project.

Where the 12 provided hypothesis patterns are insufficient, the user can access the Prolog language and program the hypothesis structure he or she wishes to use, as a Prolog procedure, then run it to get the desired retrieval. This facility is challenging for nonprogrammers, and even for programmers unfamiliar with Prolog, as is plain from the user manual examples (at least to one of us, TJR, who has been teaching Prolog for more than a decade). This is where the logic-based nature of AQUAD shows up—hypotheses simply are statements of clausal form logic embedded in Prolog procedures, along with control structures, print control statements, string search commands, and the other paraphernalia of a program; and these are what the user must write to extend the power and expressiveness of AQUAD.

Two other built-in features of AQUAD should be mentioned for their general utility in QDA work. The first is the support of qualitative matrices. The user nominates two sets of related codes, such as a range of emotions and a range of personal data on the interviewees (e.g., age group), as columns and rows. Each cell in the resulting table contains the text segments indexed by both the column and row codes for the cell, that is, the result of a Boolean intersection or AND retrieval. Inspection of the resulting matrix is a powerful heuristic in QDA.

The second feature is the *configuration analysis.* This derives from a powerful technique in formal logic, the Quine-McClusky algorithm (McDermott, 1985), which was introduced to QDA by Ragin (1987). Suppose you guess that the presence or absence of conditions C_1 to C_n may be causally relevant to the occurrence of outcome A. Then, where C_1 to C_n and A are codes that can be found in a case (e.g., interview) in AQUAD, you can use configuration analysis to see which of the C's really are relevant to the occurrence or prevention of A, whether by their presence or absence, and what those combinations of the C's are. As a simple example, hypothesize that whenever C_1, C_2, and C_3 occur in a case, so does A. And suppose your data show this is perfectly correct, but *also* show that whenever C_1 and C_2

occur but C_3 does
presence of C_1 a
occurrence of A.
all such cases au
fully this method
trieve ability to ch
tency of hypothes
elegant example o
hance precomputa
the C's and the A
of the linkage stru
or constructible in

Configuration a
of *induction* techn
gence to find nece
outcomes or, equi
of if-then rules pre
induction techniq
tion in computati
ceived, because (a
use manually; (b)
reliability in findi
of codes; (c) they a
on presence/absenc
Popperian falsific
counterinstances (r
fication of accepta
in statistical analys
way of making text
ods highly relevan

So how does lo
tional QDA? We
niques deserve m
gramming, on the
expressive, is at
complexity that m
learn it. Moreover
to express textual
code following th
and not the concept
entities (person C_1
protocol C_3 is a f
lishing social rela
doubtedly one of
coming decade is
hypotheses based
lation and not the te
formal logic to do
able in a way that is
QDA software. In t
networks we will se

An Index-Base

We turn now
ing a design a
hybrid softwa
This system
features in o

occur but C_3 does not, A still occurs. Then in the presence of C_1 and C_2, C_3 is irrelevant to the occurrence of A. Configuration analysis detects all such cases automatically. Note how powerfully this method extends the easy code-and-retrieve ability to check the correctness and consistency of hypotheses, discussed earlier. This is an elegant example of how computerization can enhance precomputational techniques. In AQUAD, the C's and the A can be not just codes, but any of the linkage structures expressible in the system or constructible in Prolog by the user.

Configuration analysis is one of several types of *induction* techniques used in artificial intelligence to find necessary or sufficient conditions of outcomes or, equivalently, to find the simplest set of if-then rules predicting a given outcome. These induction techniques deserve far more exploitation in computational QDA than they have received, because (a) they are almost impossible to use manually; (b) they are of great power and reliability in finding and simplifying associations of codes; (c) they are qualitative in nature, relying on presence/absence of codes, Occam's razor, and Popperian falsification of hypotheses by single counterinstances (rather than the relativistic quantification of acceptability of hypotheses that occurs in statistical analyses); and (d) they are a powerful way of making textual-level code-and-retrieve methods highly relevant to theorizing.

So how does logic fare as a tool in computational QDA? We have said that induction techniques deserve much development. Logic programming, on the other hand, although highly expressive, is at least currently a tool of such complexity that many users will be unwilling to learn it. Moreover, it is used entirely, at present, to express textual relations between codes (this code following that one in the text, and so on), and not the conceptual relations between the coded entities (person C_1 knows person C_2, the greeting protocol C_3 is a functional component of establishing social relationship C_4, and so on). Undoubtedly one of the research directions of the coming decade is the idea of writing and testing hypotheses based on codes at the conceptual-relation and not the textual-relation level, and using formal logic to do it, and making the logic available in a way that is habitable by the average user of QDA software. In the section below on conceptual networks we will see one partial approach to this.

An Index-Based Approach

We turn now with some trepidation to describing a design approach used in our own rather hybrid software system, NUD•IST™ version 3.0. This system combines and relates many of the features in other specialized designs described here. Like them, it is based on a code-and-retrieve facility and endeavors to go beyond simply retrieving text according to how it was coded. It can be thought of as having two major components for managing not only documents but also ideas. The first, a *document system,* holds textual-level data about documents, which may be on-line disk files or off-line documents such as books or anything else that can be sequentially segmented for coding (videotapes are supported directly via a link to the CVideo™ system). These documents may be indexed by typing in codes or by text search and autoindexing. On-line documents can be edited at any time, even after indexing. Use of multiple windows allows views into many documents or their indexing at once. Retrievals can be done by a wide range of Boolean, context, proximity, and sequencing searches, and grouped into qualitative matrices. Results of retrievals can be stored as index codes, as can the results of text searches, which can be regular expression pattern based. All NUD•IST operations can be executed in batch mode if desired, to automate repetitive work. Thus far NUD•IST is a code-and-retrieve system, and many users, we find, use it only in this way.

But the codes and references are kept in an *index system* designed also to allow the user to create and manipulate concepts and store and explore emerging ideas. The *nodes* of the index system, where indexing is kept, are optionally organized into hierarchies, or *trees,* to represent the organization of concepts into categories and subcategories, a taxonomy of concepts and index codes. Trees, of which there may be any number, are visually represented on the screen. The user can select a node, explore or change it, or move it elsewhere in the tree system. The trees and the nodes in them can represent anything the user wishes, such as people, objects, emotions, or ideas. They can store factual data (about cases, data types, settings, and so on), if nodes represent values of variables. Links between ideas, such as *causes, talked about, married,* can be represented in further nodes or in the node linkages in the trees. In this way the nodes in the index system can be treated as both textual level (coding documents) and conceptual level (recording things about the world and storing theory). This duality is made possible by the tree structure, which can represent conceptual relations, hence permitting nodes to be treated as concepts, not just as index codes into text. This is aided by being able to give nodes definitions and textual memos that the user can write and edit. (Documents can have memos too.) Document and node memos can also be treated as data documents and indexed like any other documents, so the index searching tools can be applied to them like any other documents, to explore the interrelationships of ideas being created.

Thus the index system approach builds on and extends the code-and-retrieve technique, emphasizing system closure. The user can explore the document and index systems and the relations between the two provided by the coding of documents. As theoretical-level structures change, the user can alter the index system without losing the references to documents supporting analysis at the textual level (groupings of text references by [illegible]

[illegible] out, NUD•IST adds to the node memo a log of what was done. Thus each node has a documented history, helping the user in auditing the research process as well as aiding interpretation of the index system as a structure of theoretical-level concepts and assertions. Where the index trees are used taxonomically, higher nodes automatically represent meaningful groupings of the textual data indexed at lower-level nodes, thus assembling and retrieving textual references for the generic concepts out of references for more specific ones. Techniques exploiting the tree structuring of the index system allow theory testing as well as the representation of facts and hypotheses.

As its authors, we find it easy and necessary to criticize NUD•IST: Criticisms feed back as future design features. First, NUD•IST appears, compared with the other systems described here, as a rather awkward hybrid, containing features of code-and-retrieve, ways of handling production-rule and other types of conceptual-level reasoning, conceptual representations alternative to conceptual network systems, and database storage facilities, all interacting through interlocking tools. NUD•IST was designed originally for provision of a range of software tools, from which users could choose according to their theoretical and methodological needs. We have learned that merely providing such varied tools can be confusing; tool sets must be integrated and easily accessible if they are to be used skillfully by the very wide range of researchers seeking QDA software.

Second, the system removes so many constraints, of size and variety of records and indexes, that a sort of methodological anomie can result. We have learned too that novice researchers, who may find their own rich and messy records to be alarming in their diversity, may be further alarmed by software that seems designed to celebrate diversity. User reports make it clear that the full implications of system closure are not easily grasped early. And, perhaps most important, the software offers many ways for a researcher never to finish a study. Novices too are often stalled by anxiety about creating a perfect index system up front, not trusting the promise that they can create and recreate the index system as they develop theories and discover patterns.

Third, the approach lacks visual display of conceptual-level diagrams and models, such as conceptual networks, that researchers may need in order to see their emerging theories before they can confidently continue with theory construction. They can see and manipulate visual models of their index systems, but not models of their emerging theories. They have to go to graphics programs (or even pencil and paper) to do that. Or, if their theories are of the right type, they can use conceptual network systems [illegible]

Conceptual Network Systems

Concept diagrams, conceptual graphs, semantic nets, and *conceptual networks* are all (roughly) different names given to the same idea, of representing conceptual information in a graphic manner, as opposed to production rules or the symbolic approach of formal logic. They appeal initially to researchers who have worked pictorially, by doodling diagrams on blackboards, or more formally in an attempt to give graphic representation to emerging theory and thereby draw out discovered linkages.

The objects in one's conceptual system (people, groupings of people, properties such as age or being a vegetarian, places, emotions—essentially anything one might code for a QDA project) are represented by little boxes on a sheet of paper. (Put them in alphabetical order and one has one's code list.) Now one joins various boxes with labeled arrows to indicate relationships between them; for instance, a *loves* arrow from the *John* box to the *Mary* box to indicate that John loves Mary, a *causes* arrow from the *anger* box to the *violence* box, and so on. Technically, the boxes are called *nodes,* the arrows joining them are *directed arcs,* and the whole resulting network is a *directed graph.* The arrows represent relationships and the boxes represent objects, properties, and concepts. An introduction to semantic nets, which also discusses their limitations compared with logic, may be found in Richards (1989, chap. 1); Sowa (1991) provides a full treatment. Semantic nets have also been discussed above, as the *entity-relationship approach* to designing an RDBMS.

Commonly occurring arcs are *isa* ("is a") and *ako* ("a kind of"); for example, [Mary]—isa→[Protestant]—ako→[Christian]—ako→[religious believer]. Others are *belongs-to* (class membership) and *case-of,* such as [Mary]—belongs- to→[20s]—case-of→[age group]. These often have useful logical properties that can be exploited to do reasoning about the knowledge in the graph. Ako for example is *transitive,* that is, if A ako B, and B ako C, then A ako C. Thus, above, we infer [Protestant]—ako→[religious believer]. Rather similarly, we can infer [Mary]—isa→[religious believer].

Semantic nets make for intuitive and logically rich representation systems that have, like production rules and formal logic, been widely exploited in artificial intelligence work. If one wants to see all the relationships an object in one's system has, for instance, one need only look at its node in the graph [illegible] semantic nets have their limits. One cannot use an arc to represent a three-place relation or greater, such as Reverend A married Miss B to Mr. C, or person A sold item B to person C for $D. And if one wishes to represent the fundamental logical ideas of *not, or* and *all, some,* the tricks one needs to get up to can make the graphs quite unintuitive.

Semantic nets are meant to be used with some semantic rigor, like any precise language. When a node is drawn, it is meant to be clear what that node represents (Is it a concept, the objects falling under the concept, the common property of those objects, or the collection of those objects?); when an arrow is drawn, the relationship the arrow represents is meant to be clear (What does it mean? Does it relate the two node categories as a whole or the objects categorized in the two nodes?), as is the reason for the sense of the arrow (Why not double-headed? Why not the other way? Why have it at all?).

Several researchers have recommended the use of commercial computer drawing packages for qualitative model building (e.g., Padilla, 1991), and many use such diagrams as pictorial props in their publications. But what computing, and the literature on conceptual graphs, offers is the chance to be systematic and rigorous in the construction of these graphs so that they represent knowledge that can be searched for, extracted, and reasoned with.

One systematic and advanced approach to conceptual graphs is ATLAS/ti (Mühr, 1991). Again, the basis is code-and-retrieve functions in text, and these are reasonably sophisticated, with interesting ideas, particularly the idea of being able to group codes into "families." To code-and-retrieve is added an admirable memoing facility, and codes can apply to memos as well as to the original documentary text—a system closure feature. A particularly useful retrieval idea is ordering codes by date of last use, number of references, and the like. There is a good pattern-based text search facility, the finds of which can be coded—as in HyperRESEARCH and NUD•IST, this is another system closure feature.

Conceptual graphs are supported by an on-screen "intelligent" editing facility (i.e., the system makes the drawing of nodes, arcs, and their labels trivially easy, and also makes an internal logical representation of the graph one is drawing—it is not just a picture). Nodes can be codes (and hence have associated text). Arcs can be given one of a built-in set of relation names, such as *causes, isa, part-of, contradicts,* so that when such a relation is set up, its built-in logic, such as transitivity, becomes available for the system's reasoning about the network. Alternatively, a user can choose a name of his or her own for an arc (e.g., *supports position*) and provide a logic for [illegible] ATLAS will redraw a graph if it becomes too tangled.

These graphs operate at a conceptual level, not at the textual level. That is, the relationships between the nodes (codes) relate what the nodes represent (e.g., [anger]—causes→[violence]) rather than the nodes' textual references (e.g., passages about violence follow passages about anger). Thus, like NUD•IST but in a different and more visually direct way, ATLAS represents theory and factual knowledge, not relying on its indirect representation through textual relations that might be held relevant to a conceptual linkage.

What is the value of a conceptual graph representation of a project and its information? Networks are best seen as a tremendous generalization of the rather primitive information representation available in a code-and-retrieve system, which comprises simply an unstructured set of codes plus support for exploring their textual relations. Here we have a rich representation system for using nodes not just as textual codes, but as parts of graphs modeling systems in the world being studied, theories, and so on. Allied with ATLAS's sophisticated text retrieval system, the graphs support subtle exploration of text via a visually immediate interface that relates the text to the systems or theories in the world being studied. In cases such as evidence analysis, for example, facts gleaned about the historical situation under study can be represented directly into the network, for example, [Macbeth]—killed→[Duncan], and the study of the text coded at [Macbeth] and [Duncan] provides evidence for the claim. Standard semantic network techniques (which the ATLAS system will support, although it does not seem to exploit them) even permit the relations, such as killing, to be treated as nodes and so have associated text, which adds to the richness of the represehtation and subtlety of the exploration.

So, how useful are semantic networks? They are certainly, in ATLAS, very easy to construct and rich to explore. They lack the "intelligence" of production rules or formal logic as being "runnable" theories whose execution has some definite useful result, and they (at least in the ATLAS form) lack the power to represent crucial logical concepts that a logic system has, such as expressing generality, negation or absence, and alternativeness. It is very hard to see, for example, how configuration analysis, such as in AQUAD, could be represented, let alone executed, via ATLAS semantic nets.

The type of qualitative research project involved tends to dictate whether a semantic network approach such as ATLAS will be useful. It would be much better than code-and-retrieve at data-theory bootstrapping work. Its main value is where the subject of investigation can be seen as comprising a number of topics with some major characteristics of note and relationships to each other (a system). Then it can be of considerable heuristic value to draw that up in a semantic net and use the resulting nodes and arc labels as the basis for text segmentation or other data organization. (See our remarks above about the types of projects for which the entity-relationship approach to RDBMSs applies.)

For researchers wishing to organize concepts, the tree structures of an index system approach and the semantic net modeling of theories offer related but different advantages. Trees have the advantage that their structure is uniform and easily comprehensible. But that can also be a disadvantage, as the real world is often neither uniform nor comprehensible. Semantic nets directly and visually offer more forms of concept organization. Both methods support the most common form, the taxonomic tree, in which the children of any node can be treated as specializations, in some sense, of their parent concept: a kind of, a part of, is a, member of a group, case of, and so on. These are the main types of link used in semantic nets to convey logical properties of concept relations such as transitivity, and hence to support reasoning. But semantic nets represent relations between concepts that go beyond a thesauruslike taxonomy. The trees of NUD•IST are more restricted, but NUD•IST uses index system search plus the saving of search results as nodes as an alternative to semantic nets (worse for some things—it is less visual—and better for others—the system, not the user, looks for the links, and it is flexible).

As taxonomy structures, both have limitations. Semantic nets encourage researchers to keep the number of nodes small: (a) Computer screens cannot display more than a few tens, (b) the number of links (and hence the complexity) increases with the square of the number of nodes, and (c) big networks just look confusing. ATLAS offers a number of techniques for managing that sort of growing complexity, and future research in this field must concentrate on this issue. Index trees offer a different limitation for taxonomies, given that they impose hierarchy. Some concepts can be treated as specializations of several more general ones, not just one. Allowing a node to have multiple parents could handle this, but may prove confusing. Moreover, although trees offer the user the ability to structure data and ideas about data, they do not allow him or her to name links. One purpose of naming links is to associate logical properties with the links of a given name, for example, transitivity for *a kind of.* In an index structure approach, links are recorded and explored not by dragging and naming arrows but by creating and shifting nodes, a less visual and immediate process.

Conclusions

Artificial intelligence research has thus contributed to qualitative analysis powerful techniques for managing not only [illegible] concepts, and for constructing and expressing theories. Many researchers may of course never want these features, and will use computers for enhanced code-and-retrieve for collecting related passages for their contemplation. One needs indeed to avoid the danger that the style of the software one uses can coerce a project along a particular direction.

However, coercion is not a function of sophistication—the simpler code-and-retrieve packages can coerce a project into particular directions just because of their lack of support of various analyses that can be done on retrieved codes, such as co-occurrence patterns. And it is very hard to see how features such as configuration analysis, or organization of concepts into hierarchies or nets, or indeed the very provision of conceptual-level tools, can be other than powerful heuristics for qualitative researchers—if well used. The secret is, of course, not to force a feature onto a researcher if it is not appropriate for a particular task—and to provide flexibility and a "light touch" in the more powerful features so that they do not run away with their users.

In terms of research directions, look for developments in the logic programming approach, to make its power more accessible to the nonprogrammer and to extend its rules to express more directly conceptual-level structures and knowledge about the world under study. Look for wider deployment of configuration analysis and ways of generalizing that technique, and other methods of supporting inductive searches. The entity-relationship approach needs cross-breeding with good solid code-and-retrieve facilities, to provide relational database systems more attuned to text-based QDA work. Look too for ways of supporting project management, in particular (given the tentative, cut-and-try nature of a lot of QDA exploration), support for forking research work at a point in time into several future paths, pruning the unpromising ones, backtracking to earlier forks, and pursuing more alternatives within the promising paths. The growing bridges between qualitative and quantitative research are demanding software support, so look for innovative research on how to support that computationally.

Above all, look for ways of developing computer support of conceptual-level work in text-based research, not just textual-level work. That is the very hard research, for, as we hope this chapter makes clear, although software designs imported from artificial intelligence and database research are providing the breakthroughs, none of them is exactly what QDA needs. The problem and the excitement is that QDA is probably the most subtle and intuitive of human epistemologi- [illegible] to achieve satisfactory computerization.

Appendix: Software Developers

AQUAD. G. L. Huber, Universität Tübingen, Institut für Erziehungswissenschaft I, Münzgasse 2230, 7400 Tübingen 1, Germany.

ATLAS/ti. Thomas Mühr, Technische Universität Berlin, Project Public Health A4, Hardenbergstrasse 4-5, 10623 Berlin, Germany.

CVideo. Knowledge Revolution, 15 Brush Place, San Francisco, CA 94103, USA.

The Ethnograph. Qualis Research Associates, P.O. Box 2070, Amherst, MA 01004, USA.

HyperCard 2.0. Manufactured by Apple® Corp., available from any Apple retail outlet.

HyperRESEARCH. S. Hesse-Biber, Department of Sociology, Boston College, Chestnut Hill, Boston, MA 02167, USA.

NoteCards. Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA.

NUD•IST. Qualitative Solutions and Research, Box 171, La Trobe University Post Office, Vic 3083, Australia. Fax (+61-3) 479-1441.

StorySpace. Central Services, 1703 East Michigan Avenue, Jackson, MI 49202, USA.

Note

1. Formally, if you have an if-then hypothesis of the form "If C_1 and C_2 and . . . hold, then A holds," then it is shown to be *correct* if no case is found of all the C's occurring without the A occurring; and it is shown to be *complete* if in every case where A occurs, all the C's occur too. A correct hypothesis shows that A always occurs under these conditions, whereas a complete one shows there is no other set of conditions under which A occurs—both relative to the data, of course. Plainly, code-and-retrieve methods can easily test correctness and completeness of if-then statements.

References

Agar, M. (1991). The right brain strikes back. In N. G. Fielding & R. M. Lee (Eds.), *Using computers in qualitative research* (pp. 181-194). Newbury Park, CA: Sage.

Bailey, R. W. (Ed.). (1982). *Computing in the humanities*. Amsterdam: North-Holland.

Bogdan, R. C., & Taylor, S. J. (1975). *Introduction to* [illegible] *cal approach to the social sciences*. New York: John Wiley.

Chen, P. (1976). The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems, 1,* 19-36.

Clocksin, W. F., & Mellish, C. S. (1984). *Programming in Prolog* (2nd ed.). Berlin: Springer.

Conklin, J. (1987). Hypertext: An introduction and survey. *IEEE Computer, 20,* 17-41.

Fielding, N. G., & Lee, R. M. (Eds.). (1991). *Using computers in qualitative research*. Newbury Park, CA: Sage.

Geertz, C. (1973). *The interpretation of cultures: Selected essays*. New York: Basic Books.

Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Chicago: Aldine.

Halasz, F. G., Moran, T. P., & Trigg, R. H. (1987). *NoteCards in a nutshell*. Paper presented at the ACM Conference on Human Factors in Computing Systems, Toronto.

Hammersley, M., & Atkinson, P. (1983). *Ethnography: Principles in practice*. London: Tavistock.

Hesse-Biber, S., Dupuis, P., & Kinder, T. S. (1991). HyperRESEARCH: A computer program for the analysis of qualitative data with an emphasis on hypothesis testing and multimedia analysis. *Qualitative Sociology, 14,* 289-306.

Hockey, S. (1980). *A guide to computer applications in the humanities*. London: Duckworth.

Huber, G. L., & Garcia, C. M. (1991). Computer assistance for testing hypotheses about qualitative data: The software package AQUAD 3.0. *Qualitative Sociology, 14,* 325-348.

Kirk, J., & Miller, M. L. (1986). *Reliability and validity in qualitative research*. Newbury Park, CA: Sage.

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry*. Beverly Hills, CA: Sage.

Lofland, J., & Lofland, L. (1984). *Analyzing social settings* (2nd ed.). Belmont, CA: Wadsworth.

McDermott, R. M. (1985). *Computer-aided logic design*. Indianapolis: Sams.

Miall, D. S. (Ed.). (1990). *Humanities and the computer: New directions*. Oxford: Clarendon.

Miles, M. B., & Huberman, A. M. (1984). *Qualitative data analysis: A sourcebook of new methods*. Beverly Hills, CA: Sage.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: A new sourcebook of methods* (2nd ed.). Newbury Park, CA: Sage.

Mühr, T. (1991). ATLAS/ti: A prototype for the support of text interpretation. *Qualitative Sociology, 14,* 349-371.

Padilla, R. (1991). Using computers to develop concept models of social situations. *Qualitative Sociology, 14,* 263-274.

Pfaffenberger, B. (1988). *Microcomputer applications in qualitative research.* Beverly Hills, CA: Sage.

Ragin, C. C. (1987). *The comparative method: Moving beyond qualitative and quantitative strategies.* Berkeley: University of California Press.

Richards, L., & Richards, T. J. (1991a). Computing in qualitative analysis: A healthy development? *Qualitative Health Research, 1,* 234-262.

Richards, L., & Richards, T. J. (1991b). The transformation of qualitative method: Computational paradigms and research processes. In N. G. Fielding & R. M. Lee (Eds.), *Using computers in qualitative research* (pp. 38-53). Newbury Park, CA: Sage.

Richards, L., & Richards, T. J. (in press). From filing cabinet to computer. In R. W. Burgess & A. Bryman (Eds.), *Analyzing qualitative data.* London: Routledge.

Richards, T. J. (1989). *Clausal form logic: The elements of computer reasoning systems.* London: Addison-Wesley.

Seidel, J. V., & Clark, J. A. (1984). The Ethnograph: A computer program for the analysis of qualitative data. *Qualitative Sociology, 7,* 110-125.

Shelly, A., & Sibert, G. (1985). *The QUALOG users' manual.* Syracuse, NY: Syracuse University, School of Computer and Information Science.

Sowa, J. F. (Ed.). (1991). *Principles of semantic networks: Explorations in the representation of knowledge.* San Mateo, CA: Morgan Kaufmann.

Strauss, A. L. (1987). *Qualitative analysis for social scientists.* New York: Cambridge University Press.

Strauss, A. L., & Corbin, J. (1990). *Basics of qualitative research: Grounded theory procedures and techniques.* Newbury Park, CA: Sage.

Tallerico, M. (1991). Applications of qualitative analysis software: A view from the field. *Qualitative Sociology, 14,* 275-285.

Tesch, R (1990). *Qualitative research: Analysis types and software tools.* London: Falmer.

Turner, B. A. (1981). Some practical aspects of qualitative data analysis. *Quality and Quantity, 15,* 225-247.

Weitzman, E., & Miles, M. B. (1994). *Computer programs for qualitative data analysis.* Newbury Park, CA: Sage.

Winer, L. R., & Carrière, M. (1991). A qualitative information system for data management. *Qualitative Sociology, 14,* 245-262.

DOCUM
to social
vary, an
also vari
fields wi
topic of
tionship
analysis
texts (w
research
In the
studied
terials t
of their
the begi
have bee
Kluckho
can emp
and Flor
relied u
docume
tion to t
Howeve
establis
and oth
positivi
nal wor
entific
of valid
and cor
method