Stepwise structuring: A style of life for flexible software

by ERIK SANDEWALL, STURE HÄGGLUND, CHRISTIAN GUSTAFSSON, LENNAT JONESJÖ, *and* OLA STRÖMFORS
Linköping University
Linköping, Sweden

ABSTRACT

In a life cycle perspective on software, the paper describes a strategy for initialization and successive growth of software, which emphasizes flexible introduction and flexible use. The examples in the paper are taken from office information systems or personalized data processing systems.

The key points in the paper are as follows:

1. The system should be organized so that it allows multiple representations of the same information, particularly as images *(bitmaps)*, text, and structured data.
2. New applications should first be started by using representations with relatively little structure (such as images) and only gradually shift to using more structured representations.
3. It is valuable for the end user to be able to control and make use of the gradual introduction of more structure.
4. It is useful to have software tools that facilitate the interactive work of introducing more structure into the information. Some tools that have been implemented in this project are described.

A PRACTICAL CASE: FROM TEXT FILE TO STRUCTURED DATA

When we have used office information systems in our own work, we have repeatedly found it useful to use plain text files as an interim representation before building a conventional data file. Consider a very simple example: an international address register, which contains name and address information correctly in the various formats used in different countries and which is used for one single purpose: printing adhesive mailing labels to be put on envelopes. If this address register is represented as a text file, it can be edited (with a standard text editor) and it can generate the required labels (using the standard PRINT operation).

Such an implementation leaves with the user, of course, the responsibility of making sure that all entries are correct—e.g., have the correct number of lines and observe the maximum line length. However, it is not difficult for the user to understand these requirements. If there are any mistakes, they can be observed when the labels are printed out, and the user can readily solve the problem. This arrangement has the fundamental advantage that *the user can easily master the system.* For the user, that is an advantage that is often worth the price of extra attention.

The text-editor implementation becomes impractical, of course, when the number of addresses in the directory increases and when the same information is to be used for multiple purposes. The application is then converted to a file of records in the obvious way—to a structured representation. The extra effort of converting the existing text files to the record format can be avoided if one implements the structured representation right from the start. However, it can be worth the effort, since the first stage, using the text file representation, provided a body of experience of several kinds: experience of all the odd varieties of addresses that may occur in practice (which is useful for the implementor), a familiarity with the computer system as a tool (which is significant for the end user), and finally a check of possible practical problems with computer-based solutions (such as mechanical problems with the printer and the labels).

This simple case examplifies the first two of the general principles that we proposed:

1. The data that are contained in an information processing system occupy a slot in a spectrum from less structured (in our example, the text file) to more structured (in our example, the file of records).
2. It is useful to let the early stages of software development be based on less structured data and to increase the strength of structure as the system matures. A significant advantage of this approach is that it is conservative with respect to structuring; i.e., one does not introduce more structure than necessary for processing. The organizational effects of different information media with different levels of structuring has been studied by Innis[1] and Taylor.[2]

In the remainder of the paper we shall argue that there are more than two significant points along that spectrum and discuss their character. We shall also argue an additional point:

3. Software systems for interactive information processing should support more than a single point on the structuring spectrum. In other words, data with different levels of structure should be able to coexist in the same system.

Finally, we refer to systems that have been implemented and used in our laboratory that have allowed us to develop and test these principles.

SOME OTHER EXAMPLES

Let us now discuss additional examples of applications whose data have a place along the spectrum from less structure to more structure.

References in a Text

A good example from the academic environment is the preparation of the bibliography for a scientific paper. At first the manuscripts are just text files. The first transformation is to factor out the references individually as small text files (*notices,* using the term of Sandewall et al., 1980.[3]) The main text file is changed so that it contains only the expansion command, with the file names of the text files for the various references as arguments. A preprocessor must then be used before the regular text formatter in order to reinsert the small text segments for each reference.

Even this first step is valuable because it makes it convenient for various papers to share references. A second step may be to change the text file representation of each reference to a structured representation as a record in a database. Again in this example, it is valuable to have a body of practical data at work in the intended application before the structures are decided.

From Image to Text File

We return to the problem of mixing paper-based and computer-based documents and consider an information workplace, i.e., an office or another working environment where a large number of documents are processed by people. By tradition, those documents arrive on paper.

It would be impractical to key all the contents of those documents into the computer system—i.e., to convert them to the form that we have called *text* in the previous section. Not only is it expensive to do the keypunching; it is also difficult to support all kinds of figures, tables, photographs, etc.

The obvious solution, by analogy with the argument in the previous section, is to recognize the original *image* of the arriving papers as another representation (probably implemented by raster-scan techniques), along with text and structured data. The image is of course less structured than the text: The conversion from text to image is done automatically by formatters and printout devices; the conversion from image to text is usually done manually (by key-typing), and only in some cases automatically.

The support of the image representation requires hardware as well as software. The following is a scenario for what the system could be like. The personal work station consists of a keyboard, a text screen (i.e., a conventional character display terminal), and an image screen (e.g., a full video screen). The direct user-computer dialogue is performed using the keyboard and the text screen—e.g., for issuing commands to the system. The system also contains a long-term memory for image data, using either photographic or electronic storage technology, and a short-term memory for the same kind of data (e.g., in digitized form on a disk memory).

Both kinds of image memory are kept at a central location and may be viewed from all work stations. It is well known that the technology that makes that possible practically is becoming available. Information of lasting value, such as printed reports (or reports that used to be printed), are stored in the long-term memory. In a research setting, this would also include, for example, scientific journals. Information that has just arrived and that is of general interest to the user community (bulletin board information and circulation list information) goes into short-term image memory and is transferred periodically to cheaper long-term memory. Newspaper clippings, advertisements for new products, and (in a research setting) calls for papers for conferences are examples of information that could be handled in this way.

From the perspective of the user, paper-based information that arrives in one copy to the organization is available immediately to everybody on the image screen in his or her office.

Again, the advantages of the strategy should be fairly obvious: Our everyday office life includes many documents that are, properly speaking, images, and that cannot be easily expressed as text without significant loss of information and readability. An electronic office system that is able to represent image and text side by side makes it possible to shift from one representation to the other exactly when it is worthwhile.

Redundancy in Structured Data

There is also a later step, after the conversion from text to structured data, which may be either a *normal form* representation, in the sense of database theory, or a so-called *truth maintenance system,* in the sense of artificial intelligence. If the textual stage in the conversion chain is interpreted to contain all the texts that are required by the organization and if the structured-data stage deals with structured forms of the same texts, then the next stage again should be one where the user is relieved of the duty of maintaining redundant information in cases where the same information is used in several texts. This can be achieved either by reducing the structured data to a normal form, with the standard techniques; or by keeping redundant information in the system together with operators that automatically maintain the consistency, which is what truth maintenance systems do.

The advantage of the normal-form approach is, among others, that it can store large amounts of data economically. An advantage of the truth-maintenance approach is that it can be more concretely understood by the user: the machine still contains the user's documents, and there are "demons" that propagate new information to all the relevant places. Those demons can be created and removed at will and can also be designed so that they can be asked about the reasons for their actions.

A LARGER CASE STUDY: PERSONAL PLANNING INFORMATION

Several experimental office information systems provide facilities intended to facilitate the users' personal planning: calendars, agenda lists (such as "to do" lists and tickler files), and others. Such facilities can serve a widely perceived need when they make it easier to find common time for meetings and appointments. (The potential disadvantages of making it even easier to fill up people's entire days with meetings have not been discussed as much.) Further work along these lines is envisioned: Morgan[4] points out that "in true automation, the control of when to use the tools is placed in the machine support system," and he suggests that that is a desirable goal: "Similar work at Xerox, IBM, and MIT holds much promise for truly automating in the office environment" (p. 785).

Although formal reports are hard to find, informal evidence suggests that computer-based personal planning systems do not usually become popular. We believe that the following factors contribute:

1. Many people want to be in control of how they use their time.
2. Personal planning information is needed the most by people who move around a lot. At least with today's technology, a computer terminal is not available when and where a decision is made to update a plan.
3. The computer-based system cannot compete, in terms of overall convenience, with the paper-based system of handwritten notes.[5]

This does not mean that all is well the way things are usually done. The available range of literature, courses, and tools for personal planning suggests that many people are not satisfied with how they use their own time. Some of those tools could be computer based.

Clearly, personal planning information has a structure. Some tasks such as meetings occupy a fixed location in time; others are limited by deadlines, or by requirements that things be done in a certain order. There is also a goal structure, since most tasks are intended to serve a purpose. Finally, many tasks have other information attached to them: the task of calling a person can be executed only if a phone number is available; the task of traveling to another city is associated with the information that goes into the travel expense form.

At least for a computer professional, it is tempting to diagnose that problems arise because the structure of the information is not made explicit and to implement a piece of software that will administer the information, properly structured. This would be another example of going from a less structured, paper-borne representation of the information (often implemented as a heap of paper slips, with notes scribbled on them), directly to a more structured, computer-borne representation. According to the principle that we argue in this paper, one should not attempt to do that.

In this case, the intermediary station is not computer-borne texts, but instead paper-borne structures. The following description should be interpreted as an example of what one could do, rather than as a specification. The paper-based tool, which represents many of the structures in personal planning, could be a small, looseleaf binder, with tab sheets organizing the papers in the binder into sections and subsections. There could be sections for personal calendar-style time planning (on several levels of time scale); for the schedule of the whole organization, which serves as background for the personal schedule; for agendas and deadline-directed tasks; and for the various kinds of information that are attached to the tasks and sometimes prerequisites for performing them.

Furthermore, the structure provided by the tab sheets should be further refined by a repertoire of different forms used in the binder. It is natural to have special forms for calendar sheets, address directory sheets, and agenda sheets; and the looseleaf structure would allow new forms to be introduced as significant new structures are recognized.

A structured, paper-based planning tool of this kind serves a purpose in itself, and such systems exist already in the office supply market. They are relevant to the topic of the present paper, because we argue that such a paper-based planning tool is necessary before a computer-based tool can become worthwhile. Provided that the integrity of decision making is preserved, the individual user may find it beneficial to arrange that the information in his or her planning book interacts with the information in the computer.

Interaction means that information indeed goes both ways. For example, it is clearly convenient to let the address/telephone directory in the planning book be a selective printout from a file that is shared in the organization, but it is also natural to treat the handwritten updates in one person's address/telephone printout as a source of update information for the database. Thus the interaction between paper-borne and computer-borne information should be viewed as a *paper refresh:* The user brings in a set of paper sheets with handwritten corrections, updates the information in the computer accordingly (or obtains assistance for that chore), and receives a clean set of printouts confirming that the updates have been performed.

The paper refresh operation is of course similar to how programmers work with listings of programs. It may also serve as a model for how other items of personal planning information, such as weekly schedules, communicate with the computer.

The structuring of personal planning information will then have proceeded top-down, and differently from the bottom-up structuring that we discussed above for the publication referencing. Top-down structuring aims at providing an overall structure, within which yet unstructured parts may continue to exist. For example, in an information system which supports image information in short-term memory as described above, it may be sufficient to store an image of each person's weekly plans, so that it is available for others to watch. (Dividing the plan into two columns, one of which is not publicly visible, is a natural modification.) With that design, no software can inspect or modify the contents of the week's plan—a limitation that many users will consider a distinct advantage.

SOFTWARE DEVELOPMENT STRATEGIES

We described initially how the various representations may be successive stages in the system's development process. One starts with a less structured representation and later shifts to a more structured representation when the time is ripe. The body of available information at the time is converted to the more structured form, and there is some procedure (e.g., a formatter or a report generator) that is able to recreate the less structured form from the more structured one. The more structured representation becomes the source; i.e., it is henceforth the object of successive editing.

In some cases the user may wish to keep both representations permanently. For example, ordinary business cards contain some information that it may be worthwhile to change to a more structured form in the address directory, but also some other information which is best kept as it is, such as the logotype of the company or perhaps handwritten notes on the card. The user may keep both the image of the card and the database entry in his or her OIS in such a way that one can easily go from one to the other. In this case it is unclear which of the representations should be thought of as source in the above sense.

An additional and more sophisticated case occurs when the choices of representation are used alternatingly. In an application it is often easy to find a more structured representation that accounts for most but not all of the cases. Returning to the example of the reference list in the scientific paper, most quotations fit into one of a small number of cases (book, paper in a journal, internal report, etc.); but there are also occasional references that do not fit those patterns. In the transfer from the textual representation of the individual reference to the structured representation, the user might then elect to retain the textual representation for the odd cases.

This easy way out has two drawbacks: Search operations (e.g., the search for papers with a certain author) will often not "see" the odd cases, and transformations (e.g., alternative formats for presentation of the quotation, with italics for the title of the paper, the journal, etc.) have to be done manually for the odd cases. However, those drawbacks may be easily acceptable if the volume of information is moderate and if the system is always used interactively, as is often the case in OIS. It is much worse to have to think in advance of all the cases that may possibly arise—or to deal with an inflexible system where some of the cases that should have been thought of in advance have not been.

Two general observations are that one and the same system should be able to account for the various representations of information, and that each user should be able to understand how those representations for the same information can be used and exchanged.

Comparisons with Other System Development Methods

Conventionally, software engineering has recommended a sequence of carefully separated steps (often illustrated as a staircase or waterfall) from specification of needs to operation and maintenance. Stepwise structuring recommends instead that that an initial system should be put into operation early, using general-purpose software that is able to support low levels of structuring; and that only as more experience is gained should the level of structure be increased. The disadvantage of the conventional method is, of course, that it is often difficult to understand needs and do the specifications in advance.

The method of *rapid prototyping* has been proposed repeatedly as another way of dealing with that phenomenon. It has often been quoted as a raison d'etre for various incremental programming languages, such as CS4,[6] Lisp,[7] or APL.[8] However, prototyping cannot deal with the fact that users' needs change continuously during the system's lifetime (and in fact, that the system's lifespan is often limited by its ability to adapt to these changing needs). Stepwise structuring does address that issue; but at the same time it will clearly require other kinds of software tools in order to be practical—for example, tools that support the transition to higher structuring levels.

The method of *structured growth* has been proposed by one of the present authors[7] as a strategy for the gradual extension of software. The idea there is to build an initial software system with relatively few facilities, but with an organization that supports the gradual incorporation of more and more features. Thus it is closer to stepwise structuring in character; but, whereas the method of structured growth emphasizes the gradual accumulation of more software and more variants of data structures, stepwise structuring emphasizes transformation of data between structure levels as the most significant event during the development process.

SOFTWARE TOOLS FOR STEPWISE STRUCTURING

The method of stepwise structuring formulated in this paper immediately suggests the need for a number of software tools:

1. *Support for mixed data representations.* The most important tool is an information management system (IMS) (a kind of editor) that is able to handle several kinds of data at the same time—e.g., text, structured data, figures, and images. If some data are kept on a lower structuring level, even after the bulk of the data has been transformed to a stronger structure, then this IMS will be the working tool of the computer user. But even if complete transformations are done at one time and all data are thereafter in the stronger structured form, the mixed-structure information management system is a necessary tool for those who do the conversion work.
2. *Data parsers.* Regularities of data often arise spontaneously within one structuring level; therefore a data parser can be used as a tool for increasing the structure level. For example, an address directory has fairly regular contents, even if it is stored as a text file. But of course one cannot assume that all data will fit into the presumed syntax. A data parser that is going to be used for this purpose must therefore be embedded within an IMS as just described so that it can be run under strong user control.
3. *Catalogs.* One aspect of such an IMS for mixed data is that it must include the services of a conventional file directory. The use of directories for text files is universally understood, but when one starts to use a very large number of small text files, the function of the dictionary changes from being a way of assigning mnemonic names to individual files to being a database where combinations of named objects (*entities,* in database jargon; *concepts,* in semantic network jargon) from the application domain, have *text objects* (small text files, big strings) associated with them.

Similarly, the image representation of information is practical only if it is annotated by catalogs, which, for example, will identify where a certain issue of a certain journal is stored or in which picture frames a certain article (defined by author, title, etc.) occurs, or where a certain quotation within a certain article is located. In our example, the catalog for the image information tends to merge with the database used for generating references in the bibliography in new papers. That example illustrates a general principle: We really do not need a system that is just a catalog; we need a database that is organized in terms of concepts from the application domain and that *among other things* contains what used to be catalog or directory information.

When a text or an image is annotated in a catalog or a database, some of its structure is already being identified. This suggests that the structuring operation (from image to text and from text to structured data) is often a top-down process, where one first identifies the top-level structure and decomposes the original image (or text) to a number of smaller images (or texts), which may then be again decomposed. Several of the examples follow this pattern, and a catalog in our wider sense can be viewed as the software support for top-down structuring.

There are also many examples of bottom-up structuring, where small parts of a text are broken out, but the whole text retains its character as text. The above example of the references in the research paper is a case in point, and one can add the examples of yearly activities reports, as well as curricula vitae, where lists of similar events (seminars, travel, etc.) can be structured in a bottom-up fashion.

4. *Reconversion tools.* These last examples remind us of another useful, general-purpose tool: a preprocessor for the text formatter, which takes a source-source file as input, recognizes the macro expansion commands in that file, and produces as output a source file, which may be input to the regular text formatter. For the bibliography example, the user defines a command that fetches a text file with a given name and embeds it in the main file. Later, when the individual references are upgraded from text to records, the command is redefined so that it fetches the same information from the database and annotates it appropriately (e.g., using font shifts).

   In general, there must always be a tool for reconverting data from a higher structuring level to a lower one previously used. As long as a certain structuring level is being used, one is likely to build up a number of services that make use of that level. Services that cannot be substantially improved by using the increased structure can continue to be used as they are if information with the new, higher structure is reconverted to the previously used level as a preprocessor to the service.
5. *Recognition mechanisms.* We have discussed examples where the conversion from text to structured data is done at one time. It is also frequently necessary to identify pieces of data in an incoming text in order to relate it to structured data that are already in the system—e.g., when a computer mail message contains the date and time of a forthcoming event that has to be related to the contents of the user's calendar. In general, one needs software that can recognize structure in surviving lower-level data to such an extent that they can be related to the right point in the existing structure in the system. Kofer[9] describes plans for the design of an interface between the two representations that would be adequate for this purpose.
6. *Prototype implementations.* The contents of the present paper are conclusions that we have drawn from earlier implementation efforts, namely the Linköping Office Information System (LOIS)[10,11] and the ED3 structure editor,[12] which is a candidate tool for conversion from text to structured data.

   One view of ED3 is that it is an editor for tree-structured documents, where the structure may be the one of chapters and sections inside one document, or the dictionary structure that organizes a collection of documents, or both. In an editing session, the user starts at the root of a tree and is offered a repertoire of operations for navigating and modifying the tree structure. The leaves of the tree are pieces of text and are modified by a cooperating text editor.

   ED3 has more recently been extended with support for leaves that have other types than text, particularly vector graphics and tables. As such, it illustrates the required characteristics of a dictionary that encompasses several representations, as discussed earlier in this section.

   But another view of ED3 is that it simply maintains a conventional text file together with a bracketing structure that points out the beginning and end positions in the text of blocks that may be nested recursively. When the user views a position in the ED3 tree, he/she views one selected block in the text file. The surrounding blocks are not seen at all, and in the contained blocks only the first line is seen.

   This view of ED3 is actually closer to the actual implementation. It also explains how ED3 may be useful as a tool for the transition from text to structured data: It contains commands whereby the user can conveniently bracket the text file into recursively nested blocks.

   In another project we have developed the Carousel system[11] which shows how a hierarchical information structure, similar to the one used in ED3, can be the basis of a very concise system for many of the basic services in an office information system, such as forms management and command-oriented user dialogue.

Finally, an extensible preprocessor for the formatter has been implemented within Interlisp and has been applied to a number of different uses, including the administration of reference lists. It was used for the preparation of this paper.

Work in progress includes the formal specification of an IMS that, among other things, should be a good software support environment for application development by stepwise structuring.

ACKNOWLEDGMENTS

This research was supported by the Swedish Board of Technical Development under Contract Dnr 80-3918.

REFERENCES

1. Innis, Harold A. *Empire and Communications.* Oxford: Oxford University Press, 1950.
2. Taylor, James R. "New Perspectives on the Office of the Future." In *Proceedings of the International Workshop on Office Information Systems.* Paris: INRIA, 1981.
3. Sandewall, Erik, Göran Hektor, Anders Ström, Claes Strömberg, Ola Strömfors, Henrik Sörensen, and Jaak Urmi. "Provisions for Flexibility in the Linköping Office Information System (LOIS)." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 569–577.
4. Morgan, Howard Lee. "Research and Practice in Office Automation." Invited paper. S. H. Lavington (ed), *Information Processing 80.* North-Holland, 1980.
5. Maryanski, Fred. "Guest Editor's Introduction." *Computer,* 14 (1981), p. 11.
6. Berild, Stig, and Sam Nachmens. "CS4—A Tool for Database Design by Infological Simulation." In *Proceedings of Third International Conference on Very Large Data Bases.* Published in 1977; available from IEEE Computer Society, Long Beach, California.
7. Erik Sandewall: *Programming in the Interactive Environment: The 'Lisp' Experience.* ACM Computing Surveys, Vol. 10, No. 1, pp. 35–72, March 1978.

8. Gomaa, Hassan, and Douglas B. H. Scott. "Prototyping as a Tool in the Specification of User Requirements." In *Proceedings of the 5th International Conference on Software Engineering.* New York: IEEE, 1981.
9. Kofer, G. Reinhard. *Some Software Integration Technology Concepts for Saving Money While Doing Empirical User Research.* In *Proceedings of the International Workshop on Office Information Systems.* Paris: INRIA, 1981.
10. Hägglund, Sture; other authors (names not given in report). "80-talets elektroniska kontor. Erfarenheter från LOIS-projektet." ("The Electronic Office of the 80's. Experience from the LOIS Project.") Research report LiTH-MAT-R-81-4, Software Systems Research Center, Linköping University, Sweden, 1981.
11. Sandewall, Erik. "Unified Dialogue Management in the Carousel System." In *Proceedings of the SIGACT/SIGPLAN Conference on the Principles of Programming Languages.* Albuquerque: 1982.
12. Strömfors, Ola, and Lennart Jonesjö. "The Implementation and Experience of a Structure-Oriented Text Editor." In *Proceedings of ACM SIGPLAN/SIGOA Symposium on Text Manipulation.* New York: Association for Computing Machinery, 1981.