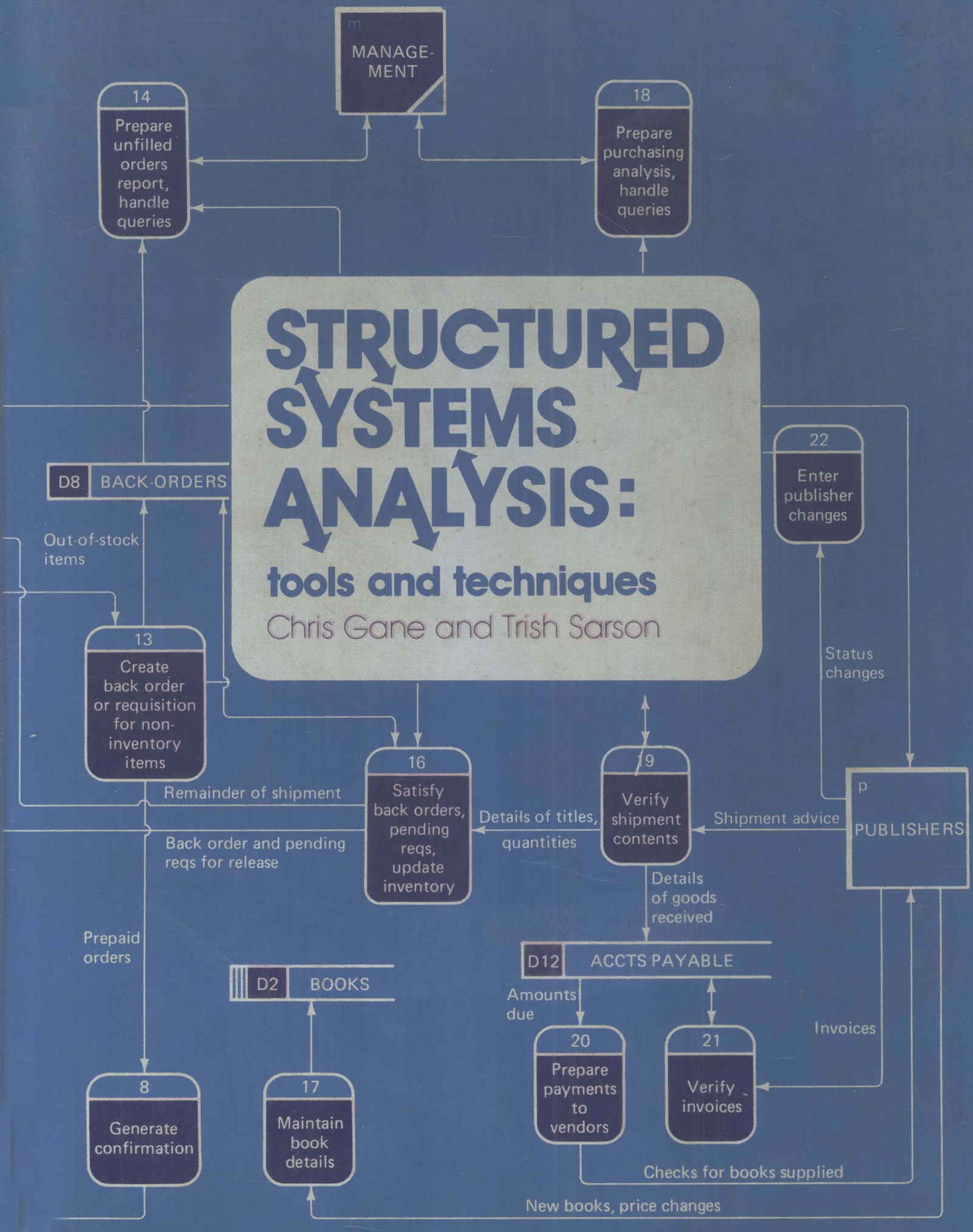



# STRUCTURED SYSTEMS ANALYSIS:

## tools and techniques

Chris Gane and Trish Sarson

# Structured Systems Analysis:
## Tools and Techniques

**Chris Gane and Trish Sarson**

**Prentice-Hall, Inc.,** Englewood Cliffs, New Jersey 07632

*Library of Congress Cataloging in Publication Data*

GANE, CHRISTOPHER P
Structured systems analysis.

Includes bibliographical references and index.
1. Electronic data processing. 2. System analysis. I. Sarson, Trish, joint author.
II. Title.
QA76.G287 1979 001.6'1 78-23173
ISBN 0-13-854547-2

Editorial/production supervision and interior design by Marianne Thomma Baltzell
Cover design by George Alon Jaediker
Manufacturing buyer: Gordon Osbourne



Printed in the United States of America

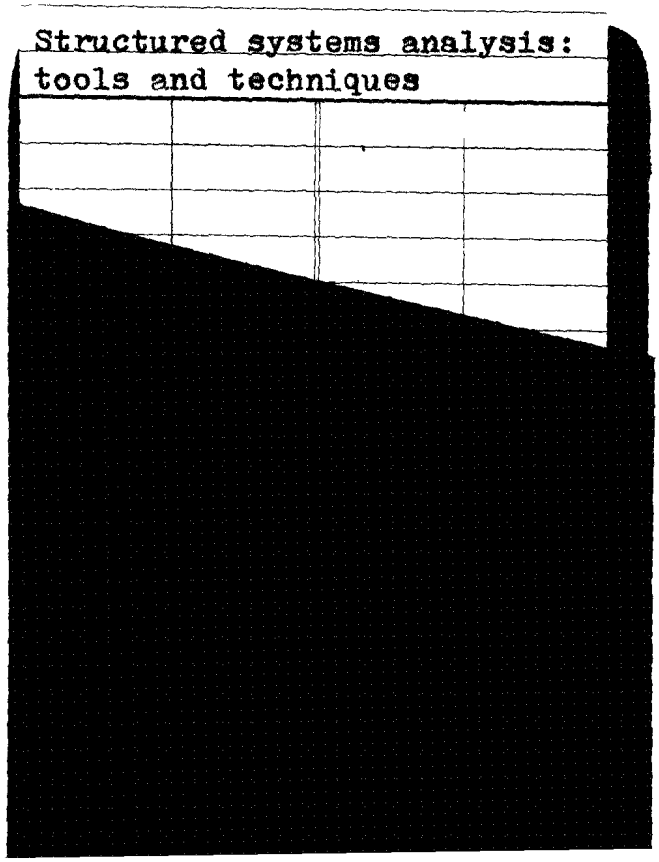
10 9 8 7 6 5 4 3

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

# Structured Systems Analysis:
## Tools and Techniques

PRENTICE-HALL SOFTWARE SERIES
Brian W. Kernighan, Advisor





# Preface

We are excited about the techniques described in this book. They are proving their worth in a troublesome area of data processing: the analysis and definition of what a new system should do if it is to be of most value to the people who are paying for it.

The discipline consists of an evolving set of tools and techniques which have grown out of the success of structured programming and structured design. The underlying concept is the building of a *logical* (non-physical) *model* of a system, using graphical techniques which enable users, analysts, and designers to get a clear and common picture of the system and how its parts fit together to meet the user's needs. Until the development of the structured systems analysis tools, there was no way of showing the underlying logical functions and requirements of a system; one very quickly got bogged down in the details of the current or proposed physical implementation.

The book starts with a discussion of some of the problems we face in analysis and then reviews the graphical tools and how they fit together to make a logical model. We then take each tool in turn and treat them in detail in Chapters 3 through 7, starting with the key tool, the logical data flow diagram. Since we are using tools which build a logical model, the approach to system development which results is somewhat different from traditional approaches; in Chapter 8 we sketch out a structured systems development methodology which takes advantage of the new tools. This methodology involves building a system top-down by *successive refinement,* first producing an overall system data flow, then developing detailed data flows, next defining the detail of data structure and process logic, then

moving into the design of a modular structure, and so on. We analyze top-down, we design top-down, we develop top-down, we test top-down. Further we recognize that good development involves *iteration;* one has to be prepared to refine the logical model and the physical design in the light of information resulting from the use of an early version of that model or design.

We distinguish the work of analysis (defining "what" the system will do) from the work of design (defining "how" it will do it), recognizing that analysts often do design and designers often do analysis. Part of the value of structured systems analysis is that it provides the designer with the inputs needed to define the programs for maximum changeability using structured design. In Chapter 9, we review the importance of changeability and the techniques and concepts of structured design, taking a realistic system, analyzing it, and designing it down to the module level.

Finally, in Chapter 10, we discuss the issues that arise in changing over to these new techniques from the traditional approaches, with their implications for management control of projects and the benefits that one can expect.

We have tried to avoid introducing new terms as far as possible; since the discipline draws on structured design (which has its own vocabulary) and relational data base theory (which has its own vocabulary), there may be some unfamiliar terminology. Each such term is explained where it first appears and is also defined in the Glossary at the end of the text.

We hope *you* will find these tools and techniques useful whether you are a systems analyst, or a designer, or a manager, or user of data processing services. We would like to hear about your experiences in using structured systems analysis, particularly if you are willing to share those experiences with others.

We gratefully acknowledge the help of those who have given us permission to reproduce their copyright material and the contributions made to the development of these ideas by our former colleagues at Yourdon, Inc., Tom de Marco, Victor Weinberg, and Ed Yourdon.

CHRIS GANE
TRISH SARSON

# Contents

# 1 The Need for Better Tools

In many ways, systems analysis is the toughest part of the development of a data processing system. It's not simply the technical difficulty of the work, though many projects demand that the analyst have deep knowledge of current DP technology. It's not simply the political difficulties that arise, especially in larger projects where the new system will serve several, possibly conflicting, interest groups. It's not simply the communication problems that arise in any situation where people of different backgrounds, with different views of the world and different vocabularies, have to work together. It's the compounding of these difficulties that makes systems analysis so hard and demanding: the fact that the analyst must play the middleman between the user community—those who have a gut-feel for their problems but find it hard to explain them and are vague about what computers can do to help—and the programming community—those who are anxious that the organization have a sharp data processing function but do not have the information to know what is best for the business. The analyst must make a match between what is currently *possible* in our onrushing technology (minis, micros, distributed processing, data base, data communications) and what is *worth doing* for the business, as run by the people in it.

Making the match in a way which is acceptable to all parties and will stand the test of time is the hardest part of the effort; if it is done well, then no matter how difficult the design and programming, the system which is built will serve the needs of the business. If it is done poorly, then no matter how excellent the implementation, the system will not be what the organization needs, and the costs will outweigh the benefits. In making that match, we need all the help we can get. This book presents some tools which have proved helpful.

The problems that the analyst faces are intertwined; that's one reason they are tough problems. We can distinguish five aspects which are worth commenting on:

*Problem 1.* The analyst finds it hard to learn enough about the business to see the system requirements through the user's eyes. (When we use the term *business,* by the way, we mean the enterprise of any organization, whether profit-making or not.) Again, and again, we hear it said, "We built a technically excellent system, but it wasn't what the users wanted." Why should this be? Why can't the analyst simply study the business and gather enough facts to specify the right system? At the heart of this problem is the fact that many user managers are "doers" rather than "explainers." They acquire and handle the information they need on an intuitive basis, without thinking in terms of information flow or decision logic. This is natural; one becomes a manager by *making* the right decision and *doing* a superior job, not necessarily by explaining how the job is done and how the decisions are made. But it means that the analyst has no right to expect a lucid explanation of the system requirements from the users; he has to help them work out their needs. At the same time, analysts do not have the gift of telepathy; they do not know what they have not been told. This painful fact shows up particularly in terms of the relative importance that users give to various features of the system. Suppose a particular manager wants a cash report each morning. Which is more important, that he have it by 8:30 even if there are some items not yet resolved, or that he have it accurate to the penny even if it takes until 11 a.m. some days? The manager knows very well and might say, "Heck, any dummy who knows anything about the business would know that!" But getting that level of intuitive feel for the trade-offs in the business is tough.

*Problem 2.* People in the user community do not yet know enough about data processing to know what is feasible and what isn't. The propaganda about computers has, in general, not left people with any specific or accurate ideas about what they can or can't do. Many people have no idea of the capability of an on-line CRT; why should they? The technology is still too young for people to have the background knowledge and exposure which would enable them to imagine the way a new system would affect them. The popular media haven't helped; the image of computers is either one of expensive and senseless mistakes or one of science fiction where boxes with a mind of their own take over the world.

Compare this situation with people's ideas about, say, the construction industry. Even though a businessman may never have commissioned a factory before, he has been in and out of factories all his working life and has formed a whole background which enables him to make sense of the things his architect will say to him. Then again, one factory is much like another; at least they will have much more in common than, say, a batch system and an on-line system. Our problems in data processing are much worse than those of the construction industry, not least because we have had no way to make a *model* of what we are going to build. In a construction or engineering

project of any size, the architect will discuss the requirements of his clients and then produce a model of what the finished structure will look like. Everyone who has an interest can look at the model, relate it to their previous experience with such structures, and form a clear idea of what they will be getting for their money. The tools of structured systems analysis enable us to produce a pictorial model of a system which can play much the same role as the model of a building or oil refinery.

*Problem 3.* The analyst can quickly get overwhelmed with detail, both the detail of the business and the technical detail of the new system. A large part of the time in the analysis phase of the project is spent in acquiring detailed information about the current situation, the clerical procedures, the input documents, the reports produced and required, the policies in effect, and the myriad of facts which are thrown up by such a complex thing as a real business. Unless there is some scheme or structure to organize these details, the analyst (or even a whole team of analysts) can become overloaded with facts and paper. The details are needed and must be available when required, but the analyst must have tools to control the detail, or he will find he "can't see the forest for the trees." Part of the value of a top-down approach, as we shall see, is that it enables one to look at the big picture and then home in on the detail of each piece as and when required.

*Problem 4.* The document setting out the details of a new system (which may be called variously the system specification, or general design, or functional specification, or some equivalent name) effectively forms a contract between the user department and the systems development group, yet it is frequently impossible for the users to understand because of its sheer bulk and the technical concepts built into it. It somewhat resembles an old-style insurance policy; the things that are really going to matter in the end are buried in the fine print. Users often make a valiant effort to master these documents and end up mentally shrugging their shoulders and signing off, saying to themselves, "Well, I guess these computer people know what they're doing." Only when the finished system is delivered do they have something which they can understand and react to, and, of course, by then it's too late.

*Problem 5.* If the specification document *can* be written in such a way as to make sense to users, it may not be very useful to the physical designers and programmers who have to build the system. Often a considerable amount of reanalysis goes on, essentially duplicating the work that the analyst has done but redefining data and process logic in terms which the programmers can use. Even if the analyst has a technical background and so writes the specification with an eye to the subsequent ease of programming, he may end up limiting the programmer's freedom of action to implement the system in the best way. The physical design of the files, programs, and input/output methods should be done by someone with up-to-date technical knowledge, based on an understanding of the complete logical requirements of the system. To begin to specify physical design before the logical model

of the system has been built is to be "prematurely physical" and too often results in an inferior design.

## 1.2 How Much Can We Blame Our Tools?

Even with the best possible analytical tools, some of the problems just discussed will always be with us. No analytical tool will enable analysts to know what is in a user's mind without being told, for instance. Nonetheless, it is the theme of this book that the problems of analysis can be significantly eased with the logical tools we describe, and we identify four limitations of our present analytical tools.

### 1.2.1 No "model" in dp

We have no way of showing a *vivid tangible model* of the system to users. It's hard for users to imagine what the new system is going to do for them until it is actually in operation, by which time it's usually too late. "How do I know what I want till I see what I get?" is the disguised cry of many users. The pictorial tools in this book give the user a better "model" of the system than was possible until now.

### 1.2.2 English narrative is too vague and long-winded

Since we have had no way of showing a tangible model, we have had to do the next best thing, which is to use English narrative to describe the proposed system. Can you imagine spending five years' salary on a custom-built house on the basis of an exhaustive narrative description of how the house will be built? No pictures, no plans, no visits to a similar house—just the 150-page narrative. "The living room, which faces south-southeast, will be 27′ × 16′ at its greatest width, with the western half taking a trapezoidal form, the west wall being 13′4″ long (abutting the northern portion of the east wall of the kitchen). . . ."

Having spent the money on the basis of the narrative and not being shown anything until the house is finished, would you be surprised if you were disappointed when you moved in? Is it surprising that users are disappointed with systems when they get them?

If you use English to describe a complex system (or building), the result takes up so much space that it's hard for the reader to grasp how the parts fit together. Worse than that, as we shall see in Chapter 5, English has some built-in problems that make it very difficult to use where precision is needed.

### 1.2.3 Flowcharts do more harm than good

If we can't make a model and English is too vague and wordy, then what about a picture? Unfortunately, up to now the only picture we have had for a system has been the flowchart. Though one flowchart *can* be worth

a thousand words, it traps the analyst into a commitment; to use the standard flowchart symbols (see Fig. 1.1) means inevitably that the analyst must commit to a *physical implementation* of the new system. The very act of drawing a flowchart means that a decision must be made as to whether the input will be on cards or through a CRT, which files will be on tape and which on disk, which programs will produce output, and so on. Yet these decisions are the essence of the *designer's* job. Once the analyst has drawn a systems flowchart, what is left for the designer to do? The designer has a choice between accepting the analyst's physical design and dealing with the details of program and file structure or (as too often happens) going back to the written specification and producing a new design from that. Neither course is satisfactory. In Fred Brooks' words,

> The manual (specification) must not only describe everything the user does see, including all interfaces; it must also refrain from describing what the user does *not* see. That is the implementer's business, and there his freedom must be unconstrained. The architect (analyst) must always be prepared to show *an* implementation for any feature he describes, but must not attempt to dictate *the* implementation. [1.1]

If the analyst and designer are the same person, drawing the flowchart must be recognized as an act of design, not of analysis. There is a great temptation to sketch a physical design of the new system before one has a full understanding of all the logical requirements; this is what is meant by being "prematurely physical."

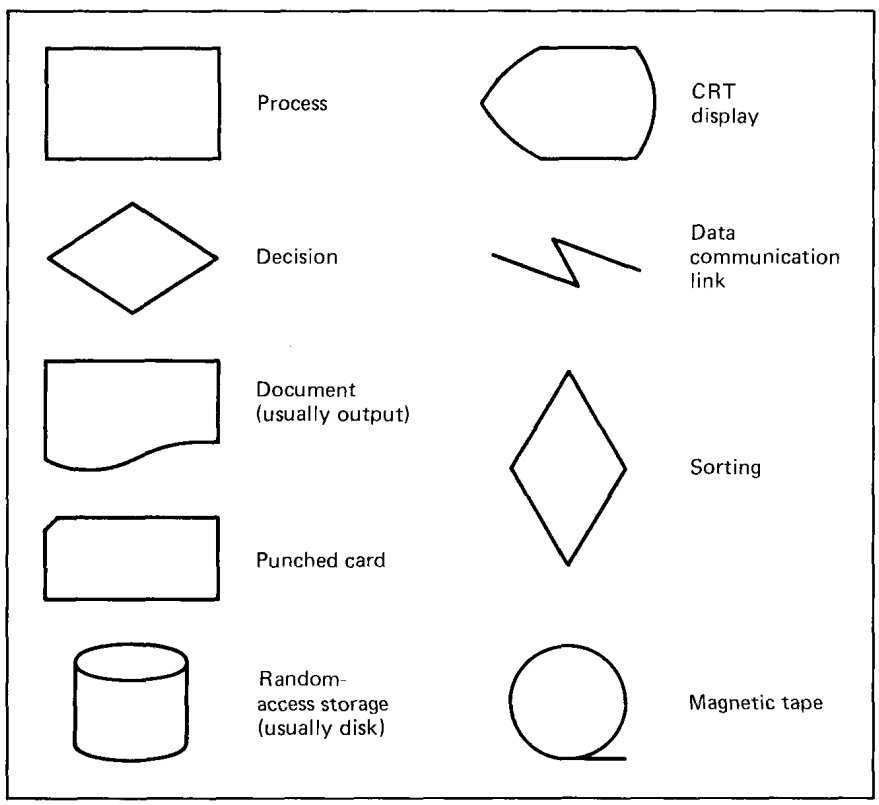



**Figure 1.1** Conventional flowchart symbols (from IBM flowcharting template X20-8020)