Computer-Aided Software Engineering
the methodologies, the products, and the future

Chris Gane

Rapid System Development Inc.

Computer-Aided Software Engineering
the methodologies, the products, and the future

Chris Gane

Rapid System Development Inc.

Prentice Hall, Englewood Cliffs, New Jersey 07632

Editorial/production supervision: bookworks
Cover design: Diane Saxe
Manufacturing buyer: Mary Ann Gloriande

Published by Prentice-Hall, Inc.
A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-176231-1

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Introduction:

What is CASE?

The acronym CASE is generally used to refer to Computer-Aided Software Engineering, though some writers have tried to extend it to mean Computer-Aided *Systems* Engineering, on the grounds that the field extends beyond the production of just software. The term was coined in the early 1980s, when it became clear that graphical tools like data flow diagrams (DFDs), entity-relationship diagrams (ERDs), and structure charts could help with systems analysis and design. Since aerospace, automobile, and other engineers got great value out of computer-aided design (CAD) systems for developing drawings and doing calculations, it seemed that computer-aided graphics might be similarly helpful to Information Systems professionals. In fact, McDonnell Douglas used their CAD expertise to produce the first product in 1981: STRADIS/DRAW.

It soon become clear, though, that a mere graphics capability was not enough; the diagram objects should be placed in a design database, which could also hold details of data elements and process logic. The system logical model built up in the design database could be tested for completeness and consistency before being printed out to form a system specification. This group of facilities was realized in EXCELERATOR, released in 1984 and described in detail in Part II of this report.

The success of EXCELERATOR really established the CASE market segment; this report describes 17 other products which essentially compete with EXCELERATOR in allowing graphical modelling of systems and the creation of a design database. As the market analysis in Chapter 10 shows, the sales of this group of products, which are referred to variously as modelling tools, analyst/designer workbenches, analysis toolkits, or front-end CASE products, are growing at a rate of some 70% per year, and will soon constitute a billion-dollar software market.

With this success, vendors of other aids to system development, such as data dictionaries, code generators, restructuring tools, and project management packages, have sought to get on the bandwagon and reposition their products as CASE products. Indeed, if software engineering is the discipline of software development and maintenance, and if their products use the computer to aid any part of those activities, they have a perfect right to do so. One may ask, however, where this process should stop. Is a debugger a CASE product? Is a test-data generator a CASE product? To an assembly language programmer, a COBOL compiler could be seen as a CASE product; after all, it uses the computer to aid in the production of software!

This report suggests that the distinguishing characteristic of a CASE product is that it builds within itself a design database, at a higher level than code statements or physical data element definitions. This design database, referred to in the report as a repository, typically holds information about the data to be stored in the system, the business logic of the processes to be implemented, the physical layout of screens and reports, and other requirements/design information. On this definition, it is not necessary to have graphical capability to be a CASE product, though most of the products reviewed here do so.

CASE products are thus a special sub-class of development/maintenance aids. They include code generators and reverse-engineering tools which extract specification-level logic from code. On this definition, programming aids such as code exercisers, debuggers, and test data generators would *not* be classed as CASE tools, since they do not build a design database; nor would restructuring tools which simply translate one set of source statements into a (more readable/changeable) set of source statements.

However the defining lines are drawn, it should be clear from this report that we are dealing with an important, dynamically-growing, new class of software which has a lot to offer in improving the speed, quality, and cost of system development and maintenance.

Structure of the report

As the Contents shows, the report is divided into three parts:

Part I: The methodologies

This section of the report presents each of the main interactive graphical and other techniques supported by some or all CASE tools, with an explanation of where the technique should be used in system development, and of how the techniques fit together.

Logical modelling of data/process
The purpose of logical modelling is to provide a reasonably rapid way for the users and designers of a system to express, exchange, and refine their initial (usually vague) ideas about its scope and content, using diagrams to show the data, the processes (functions), and their inter-relationships. Two main diagram types are used:

- data flow diagrams (DFDs), which show the processes, data stores, and data flows into, around, and out of the system. The Gane/Sarson and Yourdon/DeMarco DFD notations are presented and compared.

- entity-relationship diagrams (ERDs), which show the data entities in the system and the nature of their associations. The Martin, Chen, Ross, and LBMS notations for ERDs are presented and compared.

Other graphical techniques are discussed.

Meta-data repository
DFDs and ERDs show the relationship between data entities and process logic, but do not show the details. Each CASE product needs a place to store the details of data elements, data structures, and process logic, as well as requirements and other textual information. Physical information such as screen/report layout, database definitions, and program logic may be stored in the design database.

Data analysis - normalization
Whether or not the eventual system database will be relational or non-relational, the data structures describing each entity should be expressible in third normal form. Automatable techniques for data analysis are discussed.

Process design
Once the processes/procedures in the system have been identified, some automated aid can be provided in their detailed specification and implementation:

- Screen painting/prototyping
- Action diagrams for expressing the detailed logic to be implemented.
- Structure charts for designing invocation hierarchies of procedural programs; the Yourdon/Constantine and Jackson techniques are presented.

Code generation
Once the data structures have been designed and the logic of a process has been exactly specified, what is involved in automatic generation of code in any desired target language?

Project management
When analysts and programmers are developing systems with a CASE package, much valuable project management data can be captured as a side-benefit. The CASE package provides a shared data store of project information that can be used to record and analyze progress.

Step-by-step approaches
Two main types of step-by-step approaches to developing systems are in use: Information Engineering and Structured Systems Engineering. The report describes each one and discusses the extent to which they are converging.

Part I concludes with a brief overview of the CASE market-place and a discussion of the likely future of this type of software.

Part II: EXCELERATOR

Part II consists of a fairly detailed description of EXCELERATOR, the product whose success has, in many ways, served to establish this market segment.

Part III: Product summaries

This part of the report describes a total of 82 products from 24 vendors, including graphic modelling tools, data dictionaries, design aids, code generators and interfaces to code generators, project management aids, reverse engineering tools, and other associated products.

For each product, the report gives:

- vendor's address and phone number(s).
- the hardware/software platform(s) that support the product.
- the minimum PC configuration, where relevant.
- the first-copy price.

For each product, the report has an entry dealing with:

- the diagram types which the product supports, and whether the diagram symbols and syntax can be modified by the users.
- any significant limitations on the diagrams.
- the objects that can be stored in the Repository (design database).
- how the Repository is integrated with the graphics facility (where relevant), and how it is integrated with mainframe Repositories.
- what provision the product makes for allowing more than one user to share the Repository at a given time.
- the facility for producing reports to analyze the contents of the Repository.
- the facility for prototyping screens and reports.
- the facility for code generation.
- the facility for generating documents in various formats.
- any facility for supporting a project manager.
- any built-in assistance with design, such as an interactive dialog for normalization.
- the vendor's statement of direction.
- figures supplied by the vendor, or gleaned from industry sources, on unit volume and revenues.
- whether the vendor has a user's group.

CONTENTS

Part I

The methodologies

Chapter 1

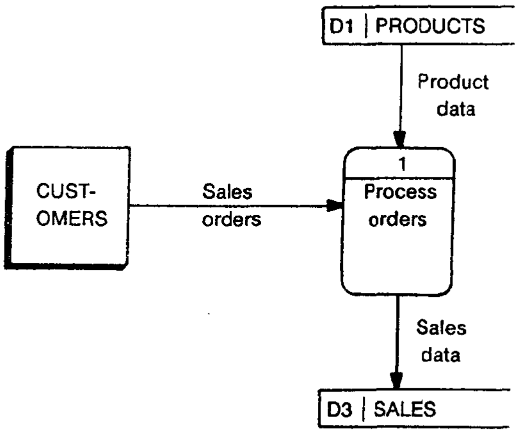
Graphical logical modelling of data/process: data flow diagrams

The purpose of a data flow diagram (DFD) is to show, for a business area or a system or part of a system, where the data comes from, where the data goes to when it leaves the system, where the data is stored, what processes transform it, and the interactions between data stores and processes.

Two principal techniques are widely used: that associated with Gane and Sarson (Ref 1-1), and that associated with Yourdon and DeMarco (Ref 1-2). The two techniques are quite similar; the differences between them are discussed in Section 1.3.

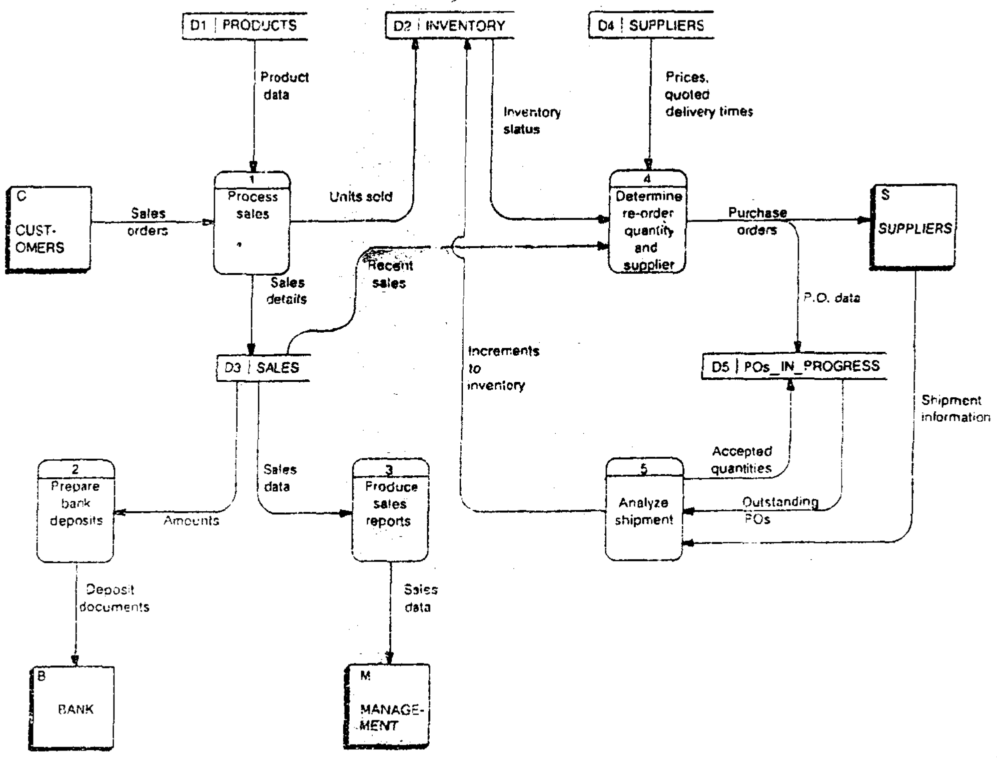
1.1 Gane/Sarson DFD technique

Consider this diagram; it shows CUSTOMERS (an external entity, something outside the system) sending in a stream of "Sales orders" along the data flow arrow.



Process 1, "Process orders," handles those orders using information from the data store of PRODUCTS (the elongated rectangle, D1), and puts information about sales into the data store named D3 SALES.

The diagram below shows the whole of the business area, using only the same four symbols. For each sale, Process 1 updates the inventory data store, D2, with the units sold. The data stored in D3 is used by Processes 2 and 3 to prepare bank deposit documents and send them to the bank and to prepare sales reports and send them to management.



At some appropriate time (notice that timing is not shown on the data flow diagram), Process 4 extracts information about the inventory status of various products and combines it with information from D3 about their past sales, to determine whether a product needs to be reordered. If so, based on information in D4, which describes the prices and delivery times quoted by suppliers, Process 4 chooses the best supplier to order from.

Purchase orders (POs) are sent out to the external entity SUPPLIERS and information about each PO is stored in D5: POS_IN_PROGRESS. When (again at some later time) a shipment is received from a supplier, Process 5 is used to analyze it, extracting data from POS_IN_PROGRESS, to see whether what has been received is what was ordered, incrementing the inventory with the accepted amount, and storing the accepted quantities in the POS_IN_PROGRESS data store.

Note the things that this (DFD) achieves:

1. The DFD sets a boundary to the area of the system and the area of the business covered by the system. Things which are represented by the external entity symbol (in this case, customers, the bank, managers, and suppliers) are, by definition, outside the system.

   Processes that are not shown on the DFD are not part of the project. For example, the diagram shows the receipt of shipments from suppliers, but not the handling of invoices received from them. This implies that "Accounts Payable" is outside the scope of the project. (This DFD is clearly not complete, by the way; there is no provision for updating PRODUCTS and SUPPLIERS, for example.)

2. The DFD is non-technical. There is nothing shown on a DFD that is not easily understandable to business people who are familiar with the business area depicted, whether or not they know anything about computers.

   Since the DFD's symbols are non-physical, it shows the underlying logical essence of the information system, and therefore is highly meaningful to business people whether or not they know anything about computers. After a minute or two's explanation of the symbols, anyone who can read a map can read a data flow.

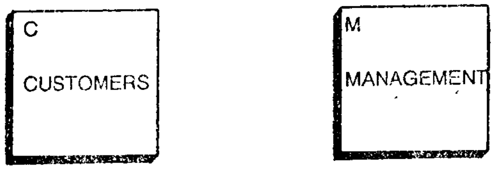
Note that the word "logical" has two meanings. It can mean "according to the rules of logic," or, as used here, it can mean "describing the underlying essence of something."

3. The DFD shows both the data stored in the system and the processes which transform that data. It shows the relationship between the data in the system and the processes in the system. (As we've noted, it doesn't show timing, but that's an important simplification.)

DFD object symbols

External entities (EEs) (also called source/sinks, source/destinations, external agents)

Gane/Sarson's conventional symbol for an EE is a square given solidity by shading on two sides. Often the external entity is given an identifying letter: M for Management, C for Customers, and so on.

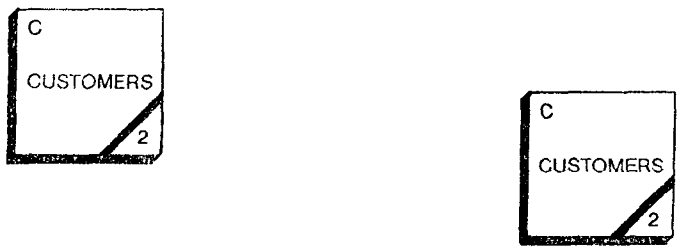


External entities are sources and/or destinations of flows of data into and out of the system. They are, by definition, outside the system under consideration. It's helpful to consider the external entity as being "behind a hole in a wall." That is to say, the system knows nothing about what is going on in the external entity. Data comes into the system through the hole in the wall; what happened to it before it came into the system does not concern us. Data from the system goes back through the hole in the wall and disappears; we are not concerned with what becomes of it. Data comes into the system only from external entities; it goes out of the system only to external entities.

If, as an analyst, you find yourself describing what goes on inside an external entity, you need to recognize that your system boundary really is wider than you are presently considering.

The external entity may be physically represented by a group of people, such as customers, or perhaps by a system, such as a payroll system. It may be just one person: the President or Comptroller.

Sometimes, for clarity, it's necessary to duplicate external entities to prevent long data flow arrows going from one side of a diagram to another. This is conventionally done by putting a diagonal stroke in the bottom right-hand corner of the external entity symbol, which says to the reader of the diagram, "There is more than one of this entity."

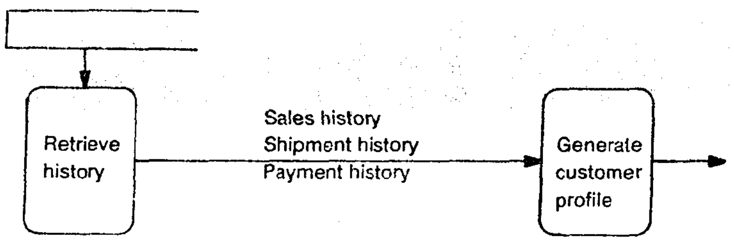


On very large diagrams, it may be convenient to put a number inside the triangle to show how many instances of the entity there are. Of course, if this is done, and then another instance of the symbol is added, all the numbers in all the instances of the symbol will need to be updated.

Data flows

Unlike an arrow on a conventional flowchart, which shows the transfer of control from one program step or module to another, the arrow on a DFD is to be thought of as a pathway, down which one or more data structures may pass as some unspecified time. The timing of the flow of data and the operation of the processes is dealt with in the specification of the processes themselves. A data flow diagram resembles a railroad map; it shows where the train tracks are laid, but it does not give the time tables.

Usually each data flow arrow has a name which describes only one data structure. Sometimes, several similar data structures may be shown passing down the same data flow, as shown here:


Retrieve history
Sales history
Shipment history
Payment history
Generate customer profile