

# COMPUTER AIDED SOFTWARE ENGINEERING

## INTRODUCTION

The concept of using the computer to help the development of computer systems and software is not new. Since the introduction of higher level languages we have had some programming tools, from the basic text editor through syntax checkers to linkers. It was only natural that as computers became more powerful, readily available and particularly with the advent of the PC, that the development of more sophisticated tools would occur.

As mechanical tools have taken the drudgery from manual tasks, so to has the computer from processing functions. The development of **C A S E** tools has been brought about by the need to improve software quality and the time taken to produce it, also the desire to automate the repetitive tasks of structured analysis and design methods. Although the term '**C A S E**' is a mid-eighties one, it's often not realised that its foundations were laid over the previous thirty years or so of analysis and design.

The subsequent evolution of **C A S E** can be thought of as a series of key innovations starting with the introduction of Third Generation languages (High Level) such as COBOL.

An important point which however, distinguishes **C A S E** from that, say of microprocessor evolution is that the new innovations did not make obsolete the previous innovations. Instead, the previous innovations tended to adapt, to exploit the new ones.

The best way to understand the true nature of **C A S E** is to view it as a 'Reservoir' of tools and techniques which have been filled at an ever increasing rate over a thirty year period.

Some key innovations which have filled this **C A S E** Reservoir are shown below, although they are not strictly in chronological order.

- 1960 High Level Languages
- Project Management
- Structured Programming
- Relational Databases
- 4GLs
- Code Generators
- Structured Methods
- Information Engineering
- Reverse Engineering
- Object Oriented
- 1990 IPSE/Repository

## THE FOUR CASE SUPER-TYPES

**C A S E** is generally divided into four categories:

**UPPER CASE:** System Definition steps, these tools include aids for Planning, Strategy, Analysis and Design.

**LOWER CASE:** System Building steps which include tools for Program definition, Code Generation, Unit and System testing.

**CASE REPOSITORY:** Often referred to as the Data Dictionary, this holds all the information gathered and cross-referenced for the other tools to access.

**LIFECYCLE CASE:** Project Management tools such as Project Scheduling and Configuration Management.

The most commonly used **C A S E** tools very often only fit into the Upper **C A S E** category with a small amount of overlap into Lower **C A S E** and **C A S E** Repository.

## CASE WORKBENCHES

These are sometimes called **CASE** Workbench Systems and are usually oriented towards the support of graphical notations such as used in the various analysis and design methods. They are either intended for the support of a specific method, such as Structured Systems Analysis and Design (SSADM), or support a range of diagram types which encompasses those used in the most common methods; these may include:

Gane & Sarson, SSADM, Jackson and Chen.

The typical components of a **C A S E** Workbench are:

- 1 A diagram editing system that is used to create data flow diagrams, structure charts, entity relationship diagrams, etc. The editor is not just a simple drafting tool but is aware of the types of entities in the diagram. It captures information about these entities and saves this information in a central repository (sometimes called a Data Dictionary).
- 2 Design analysis and checking facilities that process the design and report on errors and anomalies. As far as possible these are integrated with the editing system so that the analyst may be informed of errors during diagram creation.

- 3 Query language facilities that allow the user to browse the stored information and examine completed designs.
- 4 Data Dictionary facilities that maintain information about named entities used in a system design. Providing the facility to maintain the central store.
- 5 Report generation facilities that take information from the central store and automatically generate system documentation.
- 6 Forms generation tools that allow screen and document formats to be specified.
- 7 Import/export facilities that allow the interchange of information from the central repository with other development tools.
- 8 Some systems support skeleton code generators which generate code or code segments automatically from the design captured in the central store.

## **A WALKTHROUGH CASE METHODOLOGY**

For those new to **CASE** a simplified walkthrough the analysis and development cycle is given below:

- a) The Requirements Model: We start by trying to describe in a concise but narrative form the objectives of the proposed system along with any constraints. We also try to identify the problems with the current system (manual or computer) that is being replaced.
- b) The Data Model: We then describe the data the system must deal with using a form of network diagram. This is known as Entity Relationship Modelling and displays the major data groups (entities), how they relate to one another (relationships) and their main properties (attributes).
- c) The Process Model: We describe the main processes that will make up the system. This often takes the form of a hierarchical set of drawings, using a top-down approach. One such method commonly used is Data Flow Diagrams. These diagrams have the advantage of also showing external entities or agents, data flows and data stores as well as the processes involved.
- d) The Interaction Model: We then describe how the data and processes interact. This is often shown very clearly using a Process File Structure Chart, also referred to as a Process-Entity Matrix. This lists the entities along the X axis and the Processes down the Y axis. Each intersection is then marked to indicate whether a process creates, reads, updates or deletes a particular entity. This can be simplified to just show where a process accesses an entity.

- e) Database Design: The Database design follows naturally from the Data Model; leaving aside the issues of data normalisation and physical design, and considering the relational database approach, the entities simply become the tables, the attributes become the columns as do the relationships, shared between tables supported by referential integrity constraints.
- f) Menu Design: Again, setting aside the issues of physical design the menu design follows naturally from the Process Model with all processes in the family tree except the lowest levels becoming menu screens and choices.
- g) Process Specifications: The Process File Structure Chart (or Process-Entity Matrix) provides a good skeleton for each process in the system defining its interaction with the data. These skeletons can then be 'fleshed out' with the necessary procedural logic and error-handling in various ways including Decision Tables or Trees, Flow Charts, structured English or action diagrams, ie a form of pseudo code.
- h) Screen and Report Layouts: Together with the appropriate Process Specifications these layout form a complete specification of the screen and report programs. A number of **C A S E** tools allow screens to be defined and run in a limited way without any programming required.
- i) System Building: The next step is to build the system. This can often be achieved more quickly and more efficiently using **C A S E** tools and methodologies, as much of the work is already done.