The Use Of Computer-Aided Software Engineering Technology In Systems And Software Design

D. F. Burrows

LSI Logic Limited
1 Maidstone Road, Sidcup, Kent, England DA14 5HU

Abstract
This paper describes Computer Aided Software Engineering (CASE) Technology and its use in large-scale systems and software design projects. This paper is written in the form of a tutorial and heavy use is made of real CASE and CAD design examples. The last part of this paper presents recommendations and conclusions, and explores some future ideas about the use of CASE technology for hardware and system design as well.

Key Words: CAD, CASE, SYSTEMS, MIXED-SIGNAL

1 Introduction
This paper describes Computer Aided Software Engineering (CASE) Technology and its use in large-scale systems and software design projects. CASE Technology is to software design what CAD Technology is to hardware design. In both situations, a design methodology is implemented by a suite of tools providing for specification capture, problem analysis, design refinement and translation to a (physical) implementation. The main differences between CASE and CAD are:–

a) The level of initial design phase. CASE generally allows for much higher, more abstract levels of specification and initial design than current CAD tools do.

b) The final implementation phase. In (VLSI) CAD the final implementation phase is typically concerned with the interconnection and layout of large numbers of primitive, highly concurrent elements (eg. gates) where speed of operation and minimisation of the numbers used are prime considerations. In the CASE arena the implementation phase is typically concerned with designing and then “interconnecting” smaller numbers of more complex, highly sequential elements (eg. functions) where speed and size of the resultant code is usually less important than achieving a reliable, high quality solution.

This technical note explores the use of CASE technology with particular emphasis on its use during the development of new CAD tools. This paper is written in the form of a tutorial and heavy use is made of real CASE and CAD design examples. The last part of this paper presents recommendations and conclusions, and explores some future ideas about the use of CASE technology for hardware and system design as well.

2 General Description of CASE Technology
CASE technology comprises both a design methodology and supporting tools. The underlying design methodology provided by CASE tools is straightforward. One of the more confusing aspects is the terminology! Both methodology and terminology are described in the following sections by reviewing the CASE design flow [1,2]:–

1. Firstly, build an Essential Model ie. describe the required behaviour of the system, then
    - build an Implementation Model ie. describe automated technology organisation that embodies the required behaviour, then
    - build the System ie. embody the Implementation Model in hardware and software

2. The Use Of CASE technology in the Development of Hardware and Software
   A mixture of words and pictures are used to represent these various transformations of the original system’s requirement specification through it’s evolution to its final implementation. The following sections illustrate this by way of example of the design of a mixed signal CAD tool.

3 Essential Modelling
An Essential Model describes what a system is to do with little emphasis on how it is to do it. For CAD tool system development an Essential Model typically comprises the following parts:

3.1 A Context Diagram
The Context Diagram describes the boundary that separates the system under design from the immediate environment it exists in. Figure 2 illustrates a typical Context Diagram. In this case it is the Context Diagram for the Mixed Signal CAD system used to provide the examples throughout the rest of this paper. This Context Diagram illustrates a number of items:-

(i) The boxes represent terminators, something outside the system boundary with which the system interacts.

(ii) The lines represent Data Flows between the Terminators and the Data Transformer.

The design of the context diagram for a complex system is of significant benefit in its own right. By considering the real-world interfaces of the overall system using this, many problems that normally arise concerning the specification of a system are resolved. Furthermore, effort that is normally dissipated in trying to understand what, for example, goes on inside the box called “customer” above is, instead, used in understanding the flow of data to and from the customer, a much more tangible thing.

3.2 Transformation Schemas
The Transformation Schemas or Dataflow Diagrams describe the transformations the system makes to data in response to events. These diagrams sit hierarchically underneath the Context Diagram. The next level down Dataflow Diagram corresponding to the previous Context Diagram is illustrated in figure 2. This Dataflow Diagram illustrates a number of items:-

(i) The two lined box represents a store of data.

(ii) The bubbles still represent the data transformations the system performs.

(ii) The lines still represent Data Flows.

All the flows of the parent Context Diagram will be found to match incoming and outgoing flows on this child Dataflow Diagram, and the sum of all the flows into and out of the children of this diagram will match and so-on.

3.3 Process Specifications (P-Specs)
The leaf processes on a Dataflow Diagram have their data transformations described by Process Specifications (P-Specs), a (sometimes Pseudo-) English description of the transformations. Once again, this is best illustrated by way of example. Overleaf is the P-Spec for a leaf process bubble that handles the user input in an analogue layout editor (called ALE):–
NAME: 1/8

TITLE: Handle User Input

INPUT/OUTPUT:
- ALE-User-Entry: data_in
- ALE-Command-Table: data_in
- <Command>ALE-Character-String: data_out
- ALE-XY-Coordinate: data_out
- <Zoom_Code>ALE-Integer: data_out
- <Mouse_Entry>ALE-XY-Coordinate: data_out

BODY:
Set FINISHED to FALSE.
Repeat the following:
Read next ALE-User-Entry.

Select the policy which applies to the Entry type:

ALE-User-Entry.type is a Keypad Stroke:
Select the action from the Keypad key:
- ZOOM_IN command key:
  Set ALE_Zoom_Code to ZOOM_IN.
  Perform Zoom.
- ZOOM_OUT command key:
  Set ALE_Zoom_Code to ZOOM_OUT.
  Perform Zoom.
Default:
  Set ALE-Line_Of_Text to the string in the ALE_Keypad_Definition_List.
  Set ALE_XY_Value to NULL.
  Set FINISHED to TRUE.

ALE-User-Entry.type is a Mouse Click:
Select the action for clicked mouse key:
- LEFT:
  Set ALE_XY_Value to ALE_Cursor_Location.
  Set ALE-Line_Of_Text to NULL.
  Set FINISHED to TRUE.
- MIDDLE:
  No Action.
- RIGHT:
  Perform Pan using ALE_Cursor_Location.

ALE-User-Entry.type is a Keyboard Text Entry:
Set ALE-Line_Of_Text to ALE_User_Entry.body.
Set ALE_XY_Value to NULL.
Set FINISHED to TRUE.
Until FINISHED is TRUE.

3.4 Data Schema
The Data Schema describe the information the system must have in order to respond. Typically, data is described hierarchically in several places such as a Data Dictionary and an Entity Relationship Diagram. For example, consider the hierarchical Data Dictionary entries for a store called DF_CAD_Libraries shown in figure 2:

```
DF_CAD_Libraries (store) =
  DF_Technology_Sim_Libs
  + DF_Technology_Sim_Libs
  + DF_Technology_Tools_Libs
  + DF_Technology_Tests_Libs
```

in turn, one of these entries, DF_Technology_Sim_Libs comprises:

```
DF_Technology_Sim_Libs (data flow) =
  DF_Technology_Cell_Libs
  + DF_Technology_Sim_Libs
  + DF_Technology_Tools_Libs
  + DF_Technology_Tests_Libs
```

again, in turn, DF_Technology_Cell_Libs comprises more data and so on. The definitions usually terminate with a statement "primitive element", a Data Dictionary entry which is not comprised of any other data, but is simple and easily defined for all its users.

A simple syntax for describing entries in the Data Dictionary allows data to be optionally composed of other data, data to be replicated, items to be annotated etc. This is illustrated by the following example of the Data Dictionary entry for a control command string used on a particular CAD tool:

```
GW_Not (data flow) =
  *This is a restrictive definition of a string*
  *As a first approximation all strings must abide by these rules*

[<First>GW_Upper_Case | <First>GW_Lower_Case]
|{ [<First>GW_Upper_Case | <Rest>GW_Lower_Case ]
|<Rest>GW_Lower_Case |<Rest>GW_Upper_Case |<Rest>GW_Lower_Case |<Rest>GW_Upper_Case |<Rest>GW_Lower_Case |<Rest>GW_Upper_Case |<Rest>GW_Lower_Case |<Rest>GW_Upper_Case |
|"=" |"=" |"=" |"=" |"=" |"=" |"=" |"=" |"=" |"=" |"=" |"=" |
|7 |...
```

The relationships between many items of data can be modelled using Entity Relationship Diagrams. An example of the relationships between the data used by an FFT routine is illustrated in figure 3. The boxes represent the data, and the polygons the relationship between the data. Thus, for example, the relationship between the data sampleShapedData and the data sampleData is multiplication.

3.5 Implementation Modelling
An Implementation Model describes how a system is to perform the functions described by the Essential Model. For CAD tool system development an Implementation Model typically comprises the same types of parts that the Essential Model comprises, but with the emphasis very much on describing implementation details. Thus in an Implementation Model there are more Context Diagrams, Transformation Schemas or Dataflow Diagrams, Data Schemas (Data Dictionary and Entity Relationship Diagrams) and P-Specs. Usually the same Data Dictionary is shared between the parent Essential Model and all other related child Models.

In CAD tool development, a particularly convenient way of working has been found to be to produce an overall Essential Model describing the whole intended design environment encompassing all the current and proposed design methods and tools (parts of this model for LSI Logic's Mixed Signal Toolset has been used in the previous examples). The internal descriptions within this overall model terminate when a task is either assigned to a person or to a single tool. Then, for each new tool to be developed a separate model starting with its own Context Diagram is produced. Sometimes, a tool might have two models, its own Essential Model and then its own Implementation...
3.4 Data Schema

The Data Schema describe the information the system must have in order to respond. Typically, data is described hierarchically in several places such as a Data Dictionary and an Entity Relationship Diagram. For example, consider the hierarchical Data Dictionary entries for a store called DF_CAD_Libraries shown in figure 2:

\[
\text{DF_CAD_Libraries (store) = DF_Technology_Sim_Libs + DF_Technology_Lay_Libs + DF_Technology_Test_Libs etc.}
\]

in turn, one of these entries, DF_Technology_Lay_Libs, comprises:

\[
\text{DF_Technology_Lay_Libs (data flow) = DF_Technology_Cell_Libs + DF_Technology_Chip_Libs etc.}
\]

again, in turn, DF_Technology_Cell_Libs comprises more data and so on. The definitions usually terminate with a statement "primitive element", a Data Dictionary entry which is not comprised of any other data, but is simple and easily defined for all its users.

A simple syntax for describing entries in the Data Dictionary allows the be optionally composed of other data, data to be replicated, items to be annotated etc. This is illustrated by the following example of the Data Dictionary entry for a control command string used on a particular CAD tool:

\[
\text{GW_String (data flow) = }
\]

*This is a restrictive definition of a string*  
*As a first approximation all strings must abide by these rules*

\[
[ \{ <First>GW_Upper_Case | <First>GW_Lower_Case \} ] + [ \{ <Rest>GW_Upper_Case | <Rest>GW_Lower_Case | <Rest>GW_Numeric | = | = | | | \} ]^* .
\]

The relationships between many items of data can be modelled using Entity Relationship Diagrams. An example of the relationships between the data used by an FFT routine is illustrated in figure 3. The boxes represent the data, and the polygons the relationship between the data. Thus, for example, the relationship between the data sampleData and the data sampleData is multiplication.

3.5 Implementation Modelling

An Implementation Model describes how a system is to perform the functions described by the Essential Model. For CAD tool system development an Implementation Model typically comprises the same types of parts that the Essential Model comprises, but with the emphasis very much on describing implementation details. Thus in an Implementation Model there are more Context Diagrams, Transformation Schemas or Dataflow Diagrams, Data Schemas (Data Dictionary and Entity Relationship Diagrams) and P-Specs. Usually the same Data Dictionary is shared between the parent Essential Model and all other related child Models.

In CAD tool development, a particularly convenient way of working has been found to be produce an overall Essential Model describing the whole intended design environment encompassing all the current and proposed design methods and tools (parts of this model for LSI Logic's Mixed Signal Toolset has been used in the previous examples). The internal descriptions within this overall model terminate when a task is either assigned to a person or to a single tool. Then, for each new tool to be developed a separate model starting with its own Context Diagram is produced. Sometimes, a tool might have two models, its own Essential Model and then its own Implementation Model. However, this is debatable and depends very much upon the overall complexity of the tool, the designers experience and personal preference. Certainly, whatever is decided, each tool must have its own Implementation Model.

An example of a tool with both an Essential Model and an Implementation Model is shown in figures 4 and 5 for the FFT tool described earlier. Figure 4 illustrates part of the Essential Model describing what the tool has to do. On the other hand, the corresponding Implementation Model in figure 5 concentrates on how the tools is to be designed.

4 System Building

With all the previous Implementation Modelling and Essential Modelling in place, the last part of the design process is building the system under design. Given a CAD tool to be implemented in C, the overall picture of the code is represented by a series of hierarchical Structure Charts with the functionality of leaf elements represented by Module Specifications or M-Specs.

4.1 Structure Charts

Structure Charts use a different symbology to Datosflow Diagrams. An example of the top-level Structure Chart for an automatic layout cell ("Topocell") Builder is shown in figure 6.

The top symbol represents a hierarchical on-sheet connector, with the lower left symbol being a hierarchical off-sheet connector. The plain boxes are modules (of code-to-be). The box with a bent edges is a store. The data being passed between modules is illustrated on the flow lines.

4.2 Module Specifications (M-Specs)

This is similar to a P-Spec. The module spec can be the C code itself for the module main. In fact, any language (english, pseudo-english or any computer language) can be used to represent the description of a module. Remember, this is the specification for the final code, not necessarily the code itself.

What has been found is that it is very difficult to adopt a strict top-down approach using this method of design. Typically, once the Structured Analyst phase is complete, experimental code is written and tested and the Structured Design phase then proceeds in conjunction with coding. By the time code testing has finished the aim is to have completed the Structured Design phase as well.

5 The Tools

It is probably already apparent that a large amount of data is generated by this design method and needs to be managed by sophisticated relation database technology, supported by a good interactive graphics man-machine interface. Any attempt to use cheap, decentralised or unsophisticated CASE tools on large-scale CAD development is doomed to failure. The CASE tool market is growing very rapidly at present. LSI Logic Ltd uses one of the more sophisticated CASE tools in the market, Teamwork by Cadre at its Sidcup operation. The Teamwork database runs on a central Sun server with all the staff accessing it via there own Sun workstations. All the examples in this paper were produced from it. In general, Teamwork does the job expected of it and has been used with a reasonable degree of success by the staff at Sidcup on all the Mixed-signal CAD projects.

However, having used CASE technology for several months now, and being in the position of also writing our own CAD tools it is interesting to evaluate the maturity of the CASE market in general. If Teamwork is representative of a state-of-the-art tool then the CASE market is, in general, relatively immature compared to the (VLSI) CAD market. For example although we can do design capture and checking in Teamwork, we can not simulate the design! Such a situation would be intolerable by CAD users, yet it is apparently accepted in the CASE sector. Another loose end is the link to the implementation. In the LSI Logic's MDE™ software CAD suite, we generate layout from netlist automatically, and then at Sidcup run extensive checks on a completed design to confirm the one-to-one correspondence of the two. With Teamwork there is no way to automatically generate or check the final C source code with respect to the original Teamwork models, although it is possible to generate some C code templates from the original description. Nevertheless, Tools like Teamwork are having a major impact in the marketplace. As stated at the beginning of this section, good CASE tools are essential to support the methodology.
6 Conclusions and Recommendations

6.1 Summary of the Paper

This paper describes Computer Aided Software Engineering (CASE) Technology and its use in large-scale software design projects, with particular emphasis on its use during the development of new CAD tools.

The paper shows that there is a strong analogy between CASE Technology and CAD Technology, with the main difference being the final implementation phase.

The CASE design flow was shown to comprise the following operations:

**Essential Modelling** the what stage of design, which in turn comprises:
- A Context Diagram
- Transformation Schemas
- Process Specifications (P-Specs)
- Data Schema.

Then **Implementation Modelling**, the how stage of design, which comprises the same elements as the Essential modelling stage, but with a different emphasis.

Then **System Building**, creating the design at the code level, which in turn comprises:
- Structure Charts
- Module Specifications (M-Specs).

6.2 Recommendations

For the initial, high level design aspects of any system (comprised of hardware, software, firmware etc) CASE technology is ideal in the early stages. Where the design is to be taken to pure software (eg in CAD development) CASE can be used during the implementation phase as well, to great advantage. CAD and System designers need to understand the benefits and disadvantages of CASE technology. The whole arena of CASE technology is still immature, but will have profound effects on all aspects of future CAD and system design.

7 References


8 Acknowledgements

Many thanks are offered to all the CAD tool developers working for me at LSI Logic Ltd whose work has been used to produce the examples used in this paper.

TEAMWORK is a trademark of Cadre Technologies Incorporated.

Sun and Sun Workstation are registered trademarks of Sun Microsystems Incorporated.

MDE is a trademark of LSI Logic Corporation.
Figure 3. The Entity Relationship Diagram for an FFT routine.

Figure 4. The Essential Model of an FFT Tool.

Figure 5. The Implementation Model of an FFT Tool.

Figure 6. The Structure Chart for an automatic layout cell builder.