**UNIT-V**

**CHAPTER-I**

**COMPUTER-AIDED SOFTWARE ENGINEERING (CASE)**

computer aided software engineering (CASE) and how use of CASE tools help to improve software development effort and maintenance effort. Software is becoming the costliest component in any computer installation. Even though hardware prices keep dropping like never and falling below even the most optimistic expectations, software prices are becoming costlier due to increased manpower costs.

**CASE AND ITS SCOPE**

We first need to define what is a CASE tool and what is a CASE environment. A CASE tool is a generic term used to denote any form of automated support for software engineering. In a more restrictive sense, a CASE tool can mean any tool used to automate some activity associated with software development.

Many CASE tools are now available. Some of these tools assist in phase-related tasks such as specification, structured analysis, design, coding, testing, etc. and others to non-phase activities such as project management and configuration management.

The primary objectives in using any CASE tool are:

* To increase productivity.
* To help produce better quality software at lower cost.

**CASE ENVIRONMENT**

Although individual CASE tools are useful, the true power of a tool set can be realised only when these set of tools are integrated into a common framework or environment. If the different CASE tools are not integrated, then the data generated by one tool would have to input to the other tools.

This may also involve format conversions as the tools developed by different vendors are likely to use different formats. This results in additional effort of exporting data from one tool and importing to another. Also, many tools do not allow exporting data and maintain the data in proprietary formats.

CASE tools are characterised by the stage or stages of software development life cycle on which they focus. Since different tools covering different stages share common information, it is required that they integrate through some central repository to have a consistent view of information associated with the software.

This central repository is usually a data dictionary containing the definition of all composite and elementary data items. Through the central repository all the CASE tools in a CASE environment share common information among themselves. Thus, a CASE environment facilitates the automation of the step-by-step methodologies for software development. In contrast a CASE environment, a *programming environment* is an integrated collection of tools to support only the coding phase of software development.

The tools commonly integrated in a programming environment are a text editor, a compiler, and a debugger. The different tools are integrated to the extent that once the compiler detects an error, the editor takes automatically goes to the statements in error and the error statements are highlighted.

Examples of popular programming environments are Turbo C environment, Visual Basic, Visual C++, etc. A schematic representation of a CASE environment is shown in Figure 12.1.

The standard programming environments such as Turbo C, Visual C++, etc. come equipped with a program editor, compiler, debugger, linker, etc. All these tools are integrated. If you click on an error reported by the compiler, not only does it take you into the editor, but also takes the cursor to the specific line or statement causing the error.

 

**Beneﬁts of CASE**

Several benefits accrue from the use of a CASE environment or even isolated CASE tools.

Let us examine some of these benefits:

* A key benefit arising out of the use of a CASE environment is cost saving through all developmental phases. Different studies carry out to measure the impact of CASE, put the effort reduction between 30 per cent and 40 per cent.
* Use of CASE tools leads to considerable improvements in quality. This is mainly due to the facts that one can effortlessly iterate through the different phases of software development, and the chances of human error is considerably reduced.
* CASE tools help produce high quality and consistent documents. Since the important data relating to a software product are maintained in a central repository, redundance in the stored data is reduced, and therefore, chances of inconsistent documentation are reduced to a great extent.
* CASE tools take out most of the drudgery in a software engineer’s work. For example, they need not check meticulously the balancing of the DFDs, but can do it effortlessly through the press of a button.
* CASE tools have led to revolutionary cost saving in software maintenance efforts. This arises not only due to the tremendous value of a CASE environment in traceability and consistency checks, but also due to the systematic information capture during the various phases of software development as a result of adhering to a CASE environment.
* Introduction of a CASE environment has an impact on the style of working of a company, and makes it oriented towards the structured and orderly approach.

**CASE SUPPORT IN SOFTWARE LIFE CYCLE**

Let us examine the various types of support that CASE provides during the different phases of a software life cycle. CASE tools should support a development methodology, help enforce the same, and provide certain amount of consistency checking between different phases. Some of the possible support that CASE tools usually provide in the software development life cycle are discussed below.

**Prototyping Support**

We have already seen that prototyping is useful to understand the requirements of complex software products, to demonstrate a concept, to market new ideas, and so on.

The prototyping CASE tool’s requirements are as follows:

 Define user interaction.

 Define the system control flow.

 Store and retrieve data required by the system.

 Incorporate some processing logic.

There are several standalone prototyping tools. But a tool that integrates with the data dictionary can make use of the entries in the data dictionary, help in populating the data dictionary and ensure the consistency between the design data and the prototype. A good prototyping tool should support the following features:

 Since one of the main uses of a prototyping CASE tool is *graphical user interface* (GUI) development, a prototyping CASE tool should support the user to create aGUI using a graphics editor. The user should be allowed to define all data entryforms, menus and controls.

 It should integrate with the data dictionary of a CASE environment.

 If possible, it should be able to integrate with external user defined modules written in C or some popular high level programming languages.

 The user should be able to define the sequence of states through which a created prototype can run. The user should also be allowed to control the running of the prototype.

 The run time system of prototype should support mock up run of the actual system and management of the input and output data.

**Structured Analysis and Design**

Several diagramming techniques are used for structured analysis and structured design. CASE tool should support one or more of the structured analysis and design technique. The CASE tool should support effortlessly drawing analysis and design diagrams. The CASE tool should support drawing fairly complex diagrams and preferably through a hierarchy of levels. It should provide easy navigation through different levels and through design and analysis.

The tool must support completeness and consistency checking across the design and analysis and through all levels of analysis hierarchy. Wherever it is possible, the system should disallow any inconsistent operation, but it may be very diﬃcult to implement such a feature. Whenever there is heavy computational load while consistency checking, it should be possible to temporarily disable consistency checking.

**Code Generation**

As far as code generation is concerned, the general expectation from a CASE tool is quite low. A reasonable requirement is traceability from source file to design data. More pragmatic support expected from a CASE tool during code generation phase are the following:

 The CASE tool should support generation of module skeletons or templates in one or more popular languages. It should be possible to include copyright message, brief description of the module, author name and the date of creation in some selectable format.

 The tool should generate records, structures, class definition automatically from the contents of the data dictionary in one or more popular programming languages.

 It should generate database tables for relational database management systems.

 The tool should generate code for user interface from prototype definition for X window and MS window-based applications.

**Test CASE Generator**

The CASE tool for test case generation should have the following features:

 It should support both design and requirement testing.

 It should generate test set reports in ASCII format which can be directly imported into the test plan document.

**OTHER CHARACTERISTICS OF CASE TOOLS**

The characteristics listed in this section are not central to the functionality of CASE tools but significantly enhance the effectivity and usefulness of CASE tools

**Hardware and Environmental Requirements**

In most cases, it is the existing hardware that would place constraints upon the CASE tool selection. Thus, instead of defining hardware requirements for a CASE tool, the task at hand becomes to fit in an optimal configuration of CASE tool in the existing hardware capabilities. Therefore, we have to emphasise on selecting the most optimal CASE tool configuration for a given hardware configuration.

The heterogeneous network is one instance of distributed environment and we choose this for illustration as it is more popular due to its machine independent features. The CASE tool implementation in heterogeneous network makes use of client-server paradigm. The multiple clients which run different modules access data dictionary through this server.

The data dictionary server may support one or more projects. Though it is possible to run many servers for different projects but distributed implementation of data dictionary is not common.

The tool set is integrated through the data dictionary which supports multiple projects, multiple users working simultaneously and allows to share information between users and projects. The data dictionary provides consistent view of all project entities, e.g., a data record definition and its entity-relationship diagram be consistent. The server should depict the per-project logical view of the data dictionary. This means that it should allow backup/restore, copy, cleaning part of the data dictionary, etc.

The tool should work satisfactorily for maximum possible number of users working simultaneously. The tool should support multi-windowing environment for the users. This is important to enable the users to see more than one diagram at a time. It also facilitates navigation and switching from one part to the other.

**Documentation Support**

The deliverable documents should be organized graphically and should be able tom incorporate text and diagrams from the central repository. This helps in producing up-to date documentation. The CASE tool should integrate with one or more of the commercially available desk-top publishing packages. It should be possible to export text, graphics, tables, data dictionary reports to the DTP package in standard forms such as PostScript.

 **Project Management**

It should support collecting, storing, and analysing information on the software project’s progress such as the estimated task duration, scheduled and actual task start, completion date, dates and results of the reviews, etc.

**External Interface**

The tool should allow exchange of information for reusability of design. The information which is to be exported by the tool should be preferably in ASCII format and support open architecture. Similarly, the data dictionary should provide a programming interface to access information. It is required for integration of custom utilities, building new techniques, or populating the data dictionary.

**Reverse Engineering Support**

The tool should support generation of structure charts and data dictionaries from the existing source codes. It should populate the data dictionary from the source code. If the tool is used for re-engineering information systems, it should contain conversion tool from indexed sequential file structure, hierarchical and network database to relational database systems.

**Data Dictionary Interface**

The data dictionary interface should provide view and update access to the entities and relations stored in it. It should have print facility to obtain hard copy of the viewed screens. It should provide analysis reports like cross-referencing, impact analysis, etc. Ideally, it should support a query language to view its contents.

**Tutorial and Help**

The application of CASE tool and thereby its success depends on the users’ capability to effectively use all the features supported. Therefore, for the uninitiated users, a tutorial is very important. The tutorial should not be limited to teaching the user interface part only, but should comprehensively cover the following points:

 The tutorial should cover all techniques and facilities through logically classified sections.

 The tutorial should be supported by proper documentation.

**TOWARDS SECOND GENERATION CASE TOOL**

An important feature of the second-generation CASE tool is the direct support of any adapted methodology. This would necessitate the function of a CASE administrator for every organisation, who can tailor the CASE tool to a particular methodology. In addition, we may look forward to the following features in the second-generation CASE tool:

**Intelligent diagramming support:** The fact that diagramming techniques are useful for system analysis and design is well established. The future CASE tools would provide help to aesthetically and automatically layout the diagrams.

**Integration with implementation environment:** The CASE tools should provide integration between design and implementation.

**Data dictionary standards:** The user should be allowed to integrate many development tools into one environment. It is highly unlikely that any one vendor will be able to deliver a total solution. Moreover, a preferred tool would require tuning up for a particular system. Thus, the user would act as a system integrator. This is possible only if some standard on data dictionary emerges.

**Customisation support:** The user should be allowed to define new types of objects and connections. This facility may be used to build some special methodologies. Ideally it should be possible to specify the rules of a methodology to a *rule engine* for carrying out the necessary consistency checks.

**ARCHITECTURE OF A CASE ENVIRONMENT**

The architecture of a typical modern CASE environment is shown diagrammatically in Figure 12.2. The important components of a modern CASE environment are user interface, tool set, *object management system* (OMS), and a repository. We have already seen the characteristics of the tool set. Let us examine the other components of a CASE environment.

**User interface**

The user interface provides a consistent framework for accessing the different tools thus making it easier for the users to interact with the different tools and reducing the overhead of learning how the different tools are used.

**Object management system and repository**

Different case tools represent the software product as a set of entities such as specification, design, text data, project plan, etc. The object management system maps these logical entities into the underlying storage management system (repository).

The commercial relational database management systems are geared towards supporting large volumes of information structured as simple relatively short records. There are a few types of entities but large number of instances. By contrast, CASE tools create a large number of entities and relation types with perhaps a few instances of each.

Thus, the object management system takes care of appropriately mapping these entities into the underlying storage management system.

