9.10 CASE (COMPUTER AIDED SOFTWARE ENGINEERING)

Computer-Aided Software Engineering (CASE), in the field of Software Engineering is the scientific application of a set of tools and methods to a software system which is meant to result in high-quality, defect-free, and maintainable software products. It also refers to methods for the development of information systems together with automated tools that can be used in the software development process.

The term "Computer-aided software engineering" (CASE) can refer to the software used for the automated development of systems software, *i.e.*, computer code.

The CASE functions include analysis, design, and programming. CASE tools automate methods for designing, documenting, and producing structured computer code in the desired programming language.

Two key ideas of Computer-aided Software System Engineering (CASE) are : http://en.wikipedia.org/wiki/Computer-aided_software_engineering - cite_note- 3#cite_note-3.

The harboring of computer assistance in software development and or software maintenance processes and

An engineering approach to the software development and or maintenance.

9.11 CASE TOOLS

CASE tools are a class of software that automates many of the activities involved in various life cycle phases. For example, when establishing the functional requirements of a proposed application, prototyping tools can be used to develop graphic models of application screens to assist end users to visualize how an application will look after development. Subsequently, system designers can use automated design tools to transform the prototyped functional requirements into detailed design documents. Programmers can then use automated code generators to convert the design documents into code. Automated tools can be used collectively, as mentioned, or individually. For example, prototyping tools could be used to define application requirements that get passed to design technicians who convert the requirements into detailed designs in a traditional manner using flowcharts and narrative documents, without the assistance of automated design software.

Existing CASE Environments can be classified along four different dimensions :

- Life-Cycle Support
- Integration Dimension
- Construction Dimension
- Knowledge Based CASE dimension
- Life-Cycle Based CASE Tools

This dimension classifies CASE Tools on the basis of the activities they support in the information systems life cycle. They can be classified as Upper or Lower CASE tools.

Upper CASE Tools : Support strategic, planning and construction of conceptual level product and ignore the design aspect. They support traditional diagrammatic languages such as ER diagrams, Data flow diagram, Structure charts, Decision Trees, Decision tables, etc.

Lower CASE Tools : Concentrate on the back end activities of the software life cycle and hence support activities like physical design, debugging, construction, testing, integration of software components, maintenance, re-engineering and reverse engineering activities.

- Integration Dimension
- Three main CASE Integration dimension have been proposed
- CASE Framework
- ICASE Tools
- Integrated Project Support Environment (IPSE)

9.12 CASE ENVIRONMENTS

An environment is a collection of CASE tools and workbenches that supports the software process. CASE environments are classified based on the focus/basis of integration :

(i) Toolkits

(ii) Language-centered

(iii) Integrated

(iv) Fourth generation

(v) Process-centered

(i) Toolkits : Toolkits are loosely integrated collections of products easily extended by aggregating different tools and workbenches. Typically, the support provided by a toolkit is limited to programming, configuration management and project management. And the toolkit itself is environments extended from basic sets of operating system tools, for example, the Unix Programmer's Work Bench and the VMS VAX Set. In addition, toolkits' loose integration requires user to activate tools by explicit invocation or simple control mechanisms.

(ii) Language-centered : The environment itself is written in the programming language for which it was developed, thus enable users to reuse, customize and extend the environment. Integration of code in different languages is a major issue for language-centered environments. Lack of process and data integration is also a problem. The strengths of these environments include good level of presentation and control integration.

(iii) Integrated : These environments achieve presentation integration by providing uniform, consistent, and coherent tool and workbench interfaces. Data integration is achieved through the repository concept: they have a specialized database managing all information produced and accessed in the environment

(iv) Fourth generation : Forth generation environments were the first integrated environments. They are sets of tools and workbenches supporting the development of a specific class of program: electronic data processing and business-oriented applications. In general, they include programming tools, simple configuration management tools, document handling facilities and, sometimes, a code generator to produce code in lower level languages.