**{{Section D**

Software maintenance: The nature of maintenance, maintenance problems, maintenance techniques and tools.

Software re-engineering, reverse engineering, forward engineering: forward Engineering for Object-oriented and client/server architecture, Building blocks for CASE, CASE tools and applications.   
*}}*

-Nature of maintenance

-why maintenance required

1.client requirement.

2.organigestion change.

3.host modification

4.market conditions

-Types of maintenance.[capp]

1.corrective maintenance.

2. adaptive maintenance.

3.perfective maintenance.

4.perventive maintenance.

-Challenges and issues in maintenance.[c’ac’pc’cc’ia]

1. costs

2.adaptive changes.

3. program comprehension.

4 corrective changes.

5. impact analysis.

-Tool for software maintenance.

1. File comparator

2. compile and linker

3. debugger.

4. cross-reference generator.

5. static code analyzer.

-Software Maintenance techniques.

1. Identification & tracing

2. analysis

3 Design.

4. Implementation

5.System testing.

6 acceptance testing.

7. deliver.

8. maintenance management.

*Nature of Maintenance … What is maintenance ?*

[[2](http://swebokwiki.org/Chapter_5:_Software_Maintenance#References), c1s3]

Software maintenance stands the software product throughout its life cycle (from development to operations). Modification requests are noted and tracked, the impact of proposed changes is determined, code and other software objects are modified, testing is conducted, and a new version of the software product is released. Also, training and daily support are provided to users. The term maintainer is defined as an organization that performs maintenance activities. In this KA, the term will sometimes refer to individuals who perform those activities, contrasting them with the developers.

IEEE 14764 identifies the primary activities of software maintenance as process implementation, problem and modification analysis, modification implementation, maintenance review/acceptance, migration, and retirement. These activities are discussed in section 3.2, Maintenance Activities.

Maintainers can learn from the developers’ knowledge of the software. Contact with the developers and early involvement by the maintainer helps reduce the overall maintenance effort. In some instances, the initial developer cannot be reached or has moved on to other tasks, which creates an additional challenge for maintainers. Maintenance must take software artifacts from development (for example, code or documentation) and support them immediately, then progressively evolve/maintain them over a software life cycle.

why modifications are required??

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

* **Market Conditions**- Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
* **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
* **Host Modifications**- If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
* **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

## Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

**[Capp]**

* **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
* **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
* **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
* **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

**What is the cost of maintenance** ?

## Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

### Real-world factors affecting Maintenance Cost

* The standard age of any software is considered up to 10 to 15 years.
* Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
* As technology advances, it becomes costly to maintain old software.
* Most maintenance engineers are newbie and use trial and error method to rectify problem.
* Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
* Changes are often left undocumented which may cause more conflicts in future.

### Software-end factors affecting Maintenance Cost

* Structure of Software Program
* Programming Language
* Dependence on external environment
* Staff reliability and availability

9-7323 Vol. 4 No.01 Jan 2015 24

**What are the issues and challenges of maintenance ?**

**------- ISSUES AND CHALLENGES**

Most problems that are associated with software maintenance can be traced to deficiencies of the software development process. There are several technical and managerial problems encountered while maintaining

software [2].

[C’ac’pc’cc’ia]

A. Costs B. Impact Analysis C. Corrective Changes: D. Adaptive Changes E. Program Comprehension

1. Costs :Various research studies proposed that software maintenance consumes 60% to 80% of cost in whole

development life cycle; these surveys also report that maintenance costs are mainly due to enhancements,rather than corrections.

B. Impact Analysis :

One of the most important challenges in software maintenance is to find out the effects of a proposed

modification on the rest of the system. Impact analysis is the action of assessing the probable effects of a change with the plan of reducing sudden side effects. The task involves assessing the correctness of a

projected modification and evaluating the risks related with its completion, plus the estimates of the

effects on properties, energy and development.

C. Corrective Changes:

One of the major key issues is corrective changes because it is hard to find the correct place to do the

changes. It can be difficult to recognize the code base. If the preliminary design is reduced a minute

change might insist architecture changes that take a lot of time. If there has been a complete workaround of one problem then the next are even harder to crack. Design errors are tough to repair because it takes a lot of time and understanding of the entire code base and are linked to risks.

D. Adaptive Changes

Adaptive changes are frequently not easy due to absence of information about what the software is

being modified to. The diverse facts of the new technology to adjust to be difficult to take hold of. Also

impact analysis and discovering interfaces to the new things are difficult. Problems due to unbalanced initial design are a matter of concern.

E. Program Comprehension(understanding)

Another key issue is program ability which involves that extensive amount of time should be

expended by maintenance engineers to read and understand the code, the relevant documentation to have a better perspective on its logic, purpose and structure to maintain a part of software and to enhance the quality of software[6].

**What are the tools of software maintenance ?**

# [Tools for Software Maintenance](http://ecomputernotes.com/software-engineering/tools-for-software-maintenance)

BY DINESH THAKUR

Software maintenance involves modifying the existing software system and recording all the modifications made to it. For this, various maintenance tools are used. One of the commonly used maintenance tool is text editor. This tool creates a copy of the documentation or the code. The key feature of this tool is that it provides a medium to roll back (when required) from the current version of a file to the previous one. Several other tools used in software maintenance are listed in Table

Fc cl cg ca d

**Table Software Maintenance Tools**

|  |  |
| --- | --- |
| **Name** | **Description** |
| File comparator | Compares two files or systems and maintains the record of the differences in the files. In addition, it determines whether the two files or the systems are identical. |
| Compiler and linker | Compilers are used to check syntax errors and in some cases, locate the type of errors. When the code is compiled, the linker is used to link the code with other components, which are required for the program execution. Linkers sometimes are used to track the version numbers of the components so that appropriate versions are linked together. |
| Debugger | Allows tracing the logic of the program and examines the contents of the [registers](http://ecomputernotes.com/fundamental/input-output-and-memory/what-is-registers-function-performed-by-registers-types-of-registers) and [memory](http://ecomputernotes.com/fundamental/input-output-and-memory/what-are-the-different-types-of-ram-explain-in-detail) areas. |
| Cross-reference generator | Assures that the changes in code are in agreement with the existing code. When a change to a requirement is requested, this tool enables to know which other requirements, design, and code components will be affected. |
| Static code analyzer | Measures [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) about the code attributes such as the number of lines of code, number of spanning paths, and so on. This can be calculated when the new versions of the system are developed. |

**How many types maintenance are?**

**What are the maintenance activities ?**

## Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.

it a d i s at d mm

These activities go hand-in-hand with each of the following phase:

* **Identification & Tracing** - It involves activities relating to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
* **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
* **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
* **Implementation** - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.
* **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
* **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
* **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

* **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

**What is software reengineering ?**

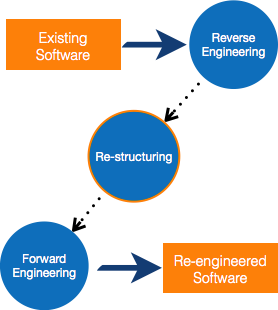
## Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



### Re-Engineering Process

* **Decide** what to re-engineer. Is it whole software or a part of it?
* **Perform** Reverse Engineering, in order to obtain specifications of existing software.
* **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
* **Re-structure data** as required.
* **Apply Forward engineering** concepts in order to get re-engineered software.

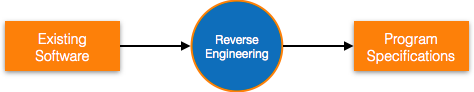
There are few important terms used in Software re-engineering

**What is software reverse engineering?**

### Reverse Engineering

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



### Program Restructuring

It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

### Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.



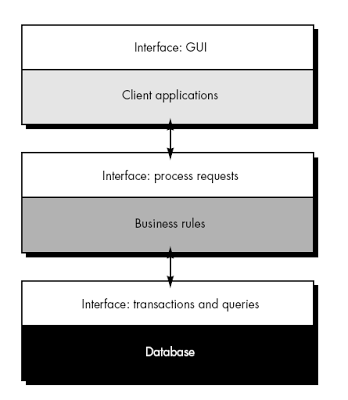
**Forward Engineering for Client/Server Architectures**

Over the past decade many mainframe applications have been reengineered to accommodate client/server architectures. In essence, centralized computing resources (including software) are distributed among many client platforms. Although a variety of different distributed environments can be designed, the typical mainframe application that is reengineered into a client/server architecture has the following features:

• Application functionality migrates to each client computer.  
• New GUI interfaces are implemented at the client sites.  
• Database functions are allocated to the server.  
• Specialized functionality (e.g., compute-intensive analysis) may remain at the server site.  
• New communications, security, archiving, and control requirements must be established at both the client and server sites.

It is important to note that the migration from mainframe to c/s computing requires both business and software reengineering. In addition, an “enterprise network infrastructure”  should be established.

Reengineering for c/s applications begins with a thorough analysis of the business environment that encompasses the existing mainframe. Three layers of abstraction  can be identified. The database sits at the foundation of a client/server architecture and manages transactions and queries from server applications. Yet these transactions and queries must be controlled within the context of a set of business rules (defined by an existing or reengineered business process). Client applications provide targeted functionality to the user community.



The functions of the existing database management system and the data architecture of the existing database must be reverse engineered as a precursor to the redesign of the database foundation layer. In some cases a new data model is created. In every case, the c/s database is reengineered to ensure that transactions are executed in a consistent manner, that all updates are performed only by authorized users, that core business rules are enforced (e.g., before a vendor record is deleted, the server ensures that no related accounts payable, contracts, or communications exist for that vendor), that queries can be accommodated efficiently, and that full archiving capability has been established.

The business rules layer represents software resident at both the client and the server. This software performs control and coordination tasks to ensure that transactions and queries between the client application and the database conform to the the established business process.

The client applications layer implements business functions that are required by specific groups of end-users. In many instances, a mainframe application is segmented into a number of smaller, reengineered desktop applications. Communication among the desktop applications (when necessary) is controlled by the business rules layer.

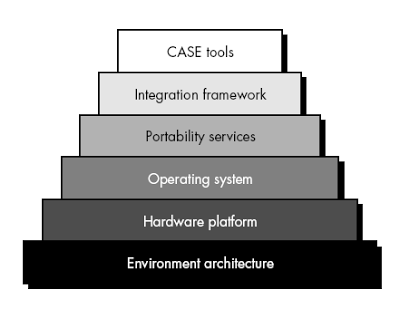
**Forward Engineering for Object-Oriented Architectures**

Object-oriented software engineering has become the development paradigm of choice for many software organizations. But what about existing applications that were developed using conventional methods? In some cases, the answer is to leave such applications “as is.” In others, older applications must be reengineered so that they can be easily integrated into large, object-oriented systems.

Reengineering conventional software into an object-oriented implementation uses many of the same techniques discussed in Part Four of this book. First, the existing software is reverse engineered so that appropriate data, functional, and behavioral models can be created. If the reengineered system extends the functionality or behavior of the original application, use-cases  are created. The data models created during reverse engineering are then used in conjunction with CRC modeling to establish the basis for the definition of classes. Class hierarchies, object-relationship models, object-behavior models, and subsystems are defined, and object-oriented design commences.

As object-oriented forward engineering progresses from analysis to design, a CBSE process model  can be invoked. If the existing application exists within a domain that is already populated by many object-oriented applications, it is likely that a robust component library exists and can be used during forward engineering. For those classes that must be engineered from scratch, it may be possible to reuse algorithms and data structures from the existing conventional application. However, these must be redesigned to conform to the object-oriented architecture

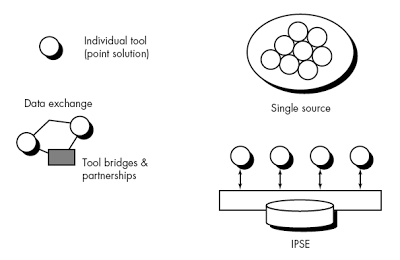
Computer aided software engineering can be as simple as a single tool that supports a specific software engineering activity or as complex as a complete "environment" that encompasses tools, a database, people, hardware, a network, operating systems, standards, and myriad other components. The building blocks for CASE are illustrated in figure. Each building block forms a foundation for the next, with tools sitting at the top of the heap. It is interesting to note that the foundation for effective CASE environments has relatively little to do with software engineering tools themselves. Rather, successful environments for software engineering are built on an environment architecture that encompasses appropriate hardware and systems software. In addition, the environment architecture must consider the human work patterns that are applied during the software engineering process.



The environment architecture, composed of the hardware platform and system support (including networking software, database management, and object management services), lays the ground work for CASE. But the CASE environment itself demands other building blocks. A set of portability services provides a bridge between CASE tools and their integration framework and the environment architecture. The integration framework is a collection of specialized programs that enables individual CASE tools to communicate with one another, to create a project database, and to exhibit the same look and feel to the end-user (the software engineer). Portability services allow CASE tools and their integration framework to migrate across different hardware platforms and operating systems without significant adaptive maintenance.

The building blocks depicted in figure above represent a comprehensive foundation for the integration of CASE tools. However, most CASE tools in use today have not been constructed using all these building blocks. In fact, some CASE tools remain "point solutions." That is, a tool is used to assist in a particular software engineering activity (e.g., analysis modeling) but does not directly communicate with other tools, is not tied into a project database, is not part of an integrated CASE environment (ICASE). Although this situation is not ideal, a CASE tool can be used quite effectively, even if it is a point solution.

The relative levels of CASE integration are shown in figure. At the low end of the integration spectrum is the individual (point solution) tool. When individual tools provide facilities for data exchange (most do), the integration level is improve slightly. Such tools produce output in a standard format that should be compatible with other tools that can read the format. In some cases, the builders of complementary CASE tools work together to form a bridge between the tools (e.g., an analysis and design tool that is coupled with a code generator). Using this approach, the synergy between the tools can produce end products that would be difficult to create using either tool separately. Single-source integration occurs when a single CASE tools vendor integrates a number of different tools and sells them as a package. Although this approach is quite effective, the closed architecture of most single-source environments precludes easy addition of tools from other vendors.



At the high end of the integration spectrum is the integrated project support environment (IPSE). Standards for each of the building blocks described previously have been created. CASE tool vendors use IPSE standards to build tools that will be compatible with the IPSE and therefore compatible with one another.

**D SOFTWARE ENGINEERING**

**CASE tool and its scope**

A CASE (Computer Aided Software Engineering) tool is a generic term used to denote any form of automated support for software engineering. In a more restrictive sense, a CASE tool means any tool used to automate some activity associated with software development. Many CASE tools are available. Some of these CASE 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.

Reasons for using CASE tools

The primary reasons for using a 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 realized only when these set of tools are integrated into a common framework or environment. CASE tools are characterized 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 development artifacts. 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 facilities the automation of the step-by-step methodologies for software development. A schematic representation of a CASE environment is shown in fig. 39.1.

*DEPT OF CSE & IT VSSUT, Burla*

Fig. 39.1: A CASE Environment

**CASE environment vs programming environment**

A CASE environment facilitates the automation of the step-by-step methodologies for software development. In contrast to a CASE environment, a programming environment is an integrated collection of tools to support only the coding phase of software development.

**Benefits of CASE**

Several benefits accrue from the use of a CASE environment or even isolated CASE tools. Some of those benefits are:

 A key benefit arising out of the use of a CASE environment is cost saving through all development phases. Different studies carry out to measure the impact of CASE put the effort reduction between 30% to 40%.

 Use of CASE tools leads to considerable improvements to 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 are considerably reduced.

*DEPT OF CSE & IT VSSUT, Burla*

 CASE tools help produce high quality and consistent documents. Since the important data relating to a software product are maintained in a central repository, redundancy in the stored data is reduced and therefore chances of inconsistent documentation is 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.

**Requirements of a prototyping CASE tool**

Prototyping is useful to understand the requirements of complex software products, to demonstrate a concept, to market new ideas, and so on. The important features of a prototyping CASE tool are as follows:

• Define user interaction

• Define the system control flow

• Store and retrieve data required by the system

• Incorporate some processing logic

**Features of a good prototyping CASE tool**

There are several stand-alone 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, prototyping CASE tool should support the user to create a GUI using a graphics editor. The user should be allowed to define all data entry forms, 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.

*DEPT OF CSE & IT VSSUT, Burla*

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

**Structured analysis and design with CASE tools**

Several diagramming techniques are used for structured analysis and structured design. The following supports might be available from CASE tools.

 A CASE tool should support one or more of the structured analysis and design techniques.

 It should support effortlessly drawing analysis and design diagrams.

 It should support drawing for fairly complex diagrams, preferably through a hierarchy of levels.

 The CASE tool should provide easy navigation through the different levels and through the design and analysis.

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

**Code generation and CASE tools**

As far as code generation is concerned, the general expectation of a CASE tool is quite low. A reasonable requirement is traceability from source file to design data. More pragmatic supports 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 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 generation CASE tool**

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.

*DEPT OF CSE & IT VSSUT, Burla*

**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, it can be emphasized on selecting the most optimal CASE tool configuration for a given hardware configuration.

The heterogeneous network is one instance of distributed environment and this can be chosen 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 who 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 sharing 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 back up/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 to 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 desktop 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 Support**

The CASE tool should support collecting, storing, and analyzing 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. *DEPT OF CSE & IT VSSUT, Burla*

**External Interface**

The CASE tool should allow exchange of information for reusability of design. The information which is to be exported by the CASE 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**

The CASE 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.

**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 organization who can tailor the CASE tool to a particular methodology. In addition, the second-generation CASE tools have following features:

 **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 lay out 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 possibly only if some standard on data dictionary emerges.

 **Customization 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.

*DEPT OF CSE & IT VSSUT, Burla*

**Architecture of a CASE environment**

The architecture of a typical modern CASE environment is shown diagrammatically in fig. 39.2. The important components of a modern CASE environment are user interface, tool set, object management system (OMS), and a repository. Characteristics of a tool set have been discussed earlier.

Fig. 39.2: Architecture of a Modern 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 (OMS) 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 such 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 entity and relation types with *DEPT OF CSE & IT VSSUT, Burla*

perhaps a few instances of each. Thus the object management system takes care of appropriately mapping into the underlying storage management system.