UNIT 3 SOFTWARE DEVELOPMENT METHODOLOGY

Structure **Page Nos.**

3.0 INTRODUCTION

During the first three decades of the computing era, the primary challenge was to develop computer hardware that reduces the cost of processing and storing data. Throughout the decade of 1980s, advances in micro electronics resulted in more computing power at an increasingly lower cost. Today, the problem is different. The primary challenge since the 1990s has been to improve the quality (and reduce the cost) of computer-based solutions – solutions that are implemented with software.

The power of the 1980s-era maintrame computer is available now from a desktop. The awesome processing and storage capabilities of modern hardware represent computing potential. Software is the mechanism that enables us to harness and tap this potential.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- know the major influencing factors on software development;

- know the process of choosing the right methodology, and
- know the features of 4GLs.

3.2 THE EVOLVING ROLE OF SOFTWARE

The context in which software has been developed is closely coupled to almost five decades of evolution of the computer system. Better hardware performance, smaller size and lower cost have yielded more sophisticated computer-based systems. We have moved from vacuum tube processor to microelectronics devices that are capable of processing 200 million instructions per second.

During the early years of computer system development, hardware underwent continual change while software was viewed by many as an afterthought. Computer programming was an art for which few systematic methods existed. Software development was virtually unmanaged – until schedules slipped or costs began to escalate. During this period, a batch orientation was used for most systems. Notable exceptions were inactive systems such as the early American Airlines reservation system and real-time defense oriented systems. For the most part however, hardware was dedicated to the execution of a single program, that, in turn, was dedicated to a specific application.

During the early years, general-purpose hardware became commonplace. Software, on the other hand, was custom designed for each application and had a relatively limited distribution. Product software (i.e., programs developed to be sold to one or more customers) was in its infancy. Most software was developed and ultimately used by the same person or organization. You wrote it, you got it running, and if it failed, you fixed it. Because job mobility was low, managers could rest assured that you would be there when bugs were encountered.

Because of this personalized software environment, design was an implicit process performed in one's head, and documentation was often nonexistent. During the early years we learned much about the implementation of computer-based systems, but relatively little about computer system engineering. In fairness, however, we must acknowledge the many outstanding computer-based systems that were developed during this era. Some of these remain in use today and provide landmark achievements that continue to justify the admiration.

The second era of computer system evolution spanned the decade from the mid-1960s to the late 1970s. Programming the multi-user systems introduced new concepts of human-machine interaction. Interactive techniques opened a new world of applications and new levels of hardware and software sophistication. Real-time systems could collect, analyze and transform data from multiple sources, thereby controlling processes and producing output in milliseconds or in a further lesser time. Advances in on-line storage led to the first generation of the database management system.

The second era was also characterized by the use of product software and the advent of software houses. Software was developed for widespread distribution in a multidisciplinary market. Programs for mainframes and minicomputers were distributed to hundreds and sometimes thousands of users.

As the number of computer-based systems grew, libraries of computer software began to expand. In-house development projects produced tens of thousands of lines of source code. Software products purchased from outside added hundreds thousands of new lines of source code. A dark cloud appeared on the horizon. All of these programs and all of these source statements had to be corrected when faults were detected,

modified as user requirements changed, or adapted to new hardware that was purchased. These activities were collectively called software maintenance. Effort spent on software maintenance began to absorb resources at an alarming rate.

Worse yet, the personalized nature of many programs made them virtually unmaintainable. A software crisis loomed on the horizon.

The third era of computer system evolution came and continues today. The distributed system – multiple computers, each performing functions concurrently and communicating with one another – greatly increased the complexity of computer-based systems. Global and local area networks, high-bandwidth digital communications, and increasing demands for 'instantaneous' data access put heavy demands on software developers.

The third era has also been characterized by the advent and widespread use of microprocessors, personal computers, and powerful desktop workstations. The microprocessor has spawned a wide array of intelligent products – from automobiles to microwave ovens, from industrial robots to blood serum diagnostic equipment. In many cases, software technology is being integrated into products by technical staff who understand hardware but are often novices in software.

The personal computer has been the catalyst for the growth of many software companies. While the software companies of the second era sold hundreds of copies of their programs, the software companies of the third era sold more than hundreds of thousands of copies. Personal computer hardware is rapidly becoming a commodity, while software provides the differentiating characteristic. In fact, as the rate of personal computer sales growth flattened during the mid-1980s, software product sales continued to grow. Many people in industry and at home spend more money on software than they did to purchase the computer on which the software would run.

The fourth era in computer software is just beginning. Object-oriented technologies are rapidly displacing more conventional software development approaches in many application areas. Authors predict that fifth-generation computers, with radically different computing architectures, and their related software will have a profound impact on the balance of political and industrial power throughout the world. Already, fourth generation techniques for software development are changing the manner in which some segments of the software community write computer programs. Expert systems and artificial intelligence software has finally moved from the laboratory into practical application for wide-ranging problems in the real world. Software based on neural networks has opened exciting possibilities for pattern recognition and human-like information processing abilities.

In the fourth era, the problems associated with computer software continue to intensify.

- Hardware sophistication has outpaced our ability to build software to tap hardware's potential.
- Our ability to maintain existing programs is threatened by poor design and inadequate resources.
- In response to these problems software engineering practices are being adopted throughout the industry.

3.3 AN INDUSTRY PERSPECTIVE

In the early days of computing, computer-based systems were developed using hardware-oriented management. Project managers focused on hardware because it was the single largest budget item for system development. To control hardware costs, managers instituted formal controls and technical standards. They demanded thorough

analysis and design before something was built. They measured the process to determine improvements that could be made.

In the early days, programming was viewed as an art. Few formal methods existed and fewer people used them. The programmer often learned his craft by trial and error. The jargon and challenges of building computer software created a mystique that few managers cared to penetrate. The software world was virtually undisciplined – and many practitioners of the day loved it.

Today, the distribution of costs for the development of computer-based systems has changed dramatically. Software, rather than hardware, is often the largest single cost item. For the past decade managers and many technical practitioners have asked the following questions:

- Why does it take so long to get programs developed?
- Why are costs so high?
- Why can't we find all errors before we give the software to our customers?
- Why do we have difficulty in measuring progress as software is being developed.

These, and many other questions, are a manifestation of the concern about software and the manner in which it is developed, a concern that has led to the adoption of software engineering practices.

The efforts that go into the design and development of computerized applications are enormous: detailed studies need to be done for analyzing the minute nuances of the manual system, defining the functions of the new system, designing coding, testing and finally implementing it.

But, if we look closely at any computerisation exercise, we find that the same set of activities need to be performed even if the system is a run-of-the-mall. Of course, when the system is implemented, the end-users invariably find certain things not happening the way they want, or find errors and bugs. So, the software enters the maintenance phase when these problems are tackled by the developers. These activities, from Problem Definition to Implementation and Evaluation, constitute the so-called Software Development Life Cycle (SDLC).

The expectations of end-users from these miracle machines have multiplied manifold. With so much hype surrounding computers and their capabilities, users are often not clear on what to expect from computers and how soon. Thus, if the computerized system fails to perform as per their dreams, then the users get disillusioned with it very soon! Also, with their penchant for jargon, the systems personnel are often unable to communicate properly with the end-users and rarely come to a mutual understanding of the problem on hand and the deliverables expected.

With the task defined being ambiguous, it is no wonder that neither of the parties agrees upon the final product and the result is the endless maintenance phase that leads to cost and time overruns that computerisation projects are notorious for. Fire-fighting takes centre-stage. Documentations, if any, never keeps pace with the system.

3.4 SOME INITIAL SOLUTIONS

One of the early solutions proposed for these problems was Structured Programming. But, soon it was found to be inadequate since merely writing programs in a more disciplined way did not seem to improve things drastically. The key apparently lay in having a more holistic picture of the problem: giving enough thought to the way the program modules are interfaced with each other; and in minimizing the effect of

changes in one module on other modules. This gave birth to Structured Design with its principles of top-down design, minimum coupling between modules, maximum cohesion within the module, and so on.

But it still did not help if you have an excellent solution that addresses the wrong problem! Or for a problem that is itself ill understood. The real solution, hence, was found in Structured Analysis, which was to come first in the chain of interrelated activities of the SDLC. This would help in understanding the existing system well and defining users' problems and requirements. Thus, having clearly understood the 'What' of the problem, Structured Design would look for the 'How' of the solution and Structured Programming would carry out this sound design.

Along with these principles came a host of techniques like the ones for modeling the system's functions, understanding the relationships between the system's data groups, designing optimal data groups, designing optimal program modules, etc. Used in isolation with the above limited objectives, this set of discrete tools again fell short somewhere, and the search continued.

3.5 STRUCTURED METHODOLOGIES

The next quantum jump came in the form of integrated structured methodologies which specify:

- A set of activities that are to be carried out in developing an information system;
- The sequence and the interrelationships of these activities;
- The tools and techniques that will be used for accomplishing these activities, and the way to record the results of these tasks.

The proposed system is detailed as a Structured Specification and has the properties of being graphical, top-down partitionable (moving from an interview to the details), and clearly separates the physical model from the logical model. This specification forms the basis for the Structured Design phase and the Program Specifications derived at the end of this later phase lead to the coding phase and so on.

A methodology integrates the activities of SDLC. The completion of the earlier phase is a precondition to the start of the later phases. Of course, some amount of controlled iteration is also allowed within or across phases in order to correct errors or to accommodate changes owing to better understanding of the system as the cycle proceeds.

Thus, structured methodologies go well beyond merely clubbing a set of tools. They have a well defined scope coverage, prescribing standards and conventions and providing a reference framework applicable to all projects. They enforce simultaneous documentation and define the deliverables from the system clearly. Estimates can be made easily, and a clear work breakdown structure facilitates Project Management.

Structured methodologies are thus beneficial to a whole host of constituents like the management, end-users, systems management, analysts, programmers and quality assurance personnel.

3.6 MAJOR INFLUENCING FACTORS

The following are some of the recent developments that influence the way methodologies are being looked at :

1) Evolution of End-user computing

2) Emergence of CASE tools

3) Use of Prototyping and 4GL Tools

4) Relational Databases

5) Object Oriented Programming

6) Graphical User Interfaces.

3.6.1 Evolution of End-User Computing

This has brought in a mixture of good and bad influences on systemization of the development process. Since end-users are now more familiar with computer technology and have first-hand exposure to it, they are more aware of its potential and its limitations. They can be involved more in the systems development process and interact more fruitfully with systems personnel. The importance of documentation may again be lost.

3.6.2 Emergence of CASE Tools

As in other application areas, more and more work in systems development is also being transferred to the computer itself, while the human being retains only the control element. Computer Aided Software Engineering (CASE) tools are based on the above philosophy and some of their features are: Diagramming support, screen painting, data dictionary maintenance, documentation support, etc. Certain CASE tools provide methodology-specific support and many of them are multi-user tools.

Some advantages of CASE tools are: Integration of the activities of SDLC, automatic standardization, guaranteed correctness and level of quality, removal of monotony, self-documentation, reduce cost and time of development, etc.

RDBMS Engineers use the CASE System to:

- Assist them in gathering the initial requirements from end-user
- Analyze these requirements and determine their feasibility
- Design the system's general algorithms.
- Design an actual detailed implementation in terms of the target environment (hardware and operating system, specific RDBMS, etc.)
- Check their designs for completeness and consistency, and for contravention of specific RDBMS naming conventions.
- Automatically generate the RDBMS (tables, indices and forms) from the design.
- Maintain their existing system by reverse engineering the original databases from their host machines.
- Control their development efforts through the medium of our configuration management tools.

CASE methods employ the structured approach to software engineering and comprise various methods in which one draws diagrams or models of the computer system to be built. Each model portrays a certain aspect of the system, with four views required to adequately model a system that uses and RDBMS as the data repository. These four views are: Data Flow Diagrams (DFSs), Entity Relationship Diagrams (ERDs), Program Structure Diagrams (PSDs), and Form/Report templates.

Some disadvantages of CASE Tools could be: Reinforcement of the tendency of systems personnel to work only with the machine (and not with human beings, e.g., users in Analysis phase, or colleagues in Design phase); loss of data due to improper

security or corruption, high cost for reasonable level of sophistication, etc. There may be a belief that the CASE tool is a panacea for all ills.

3.6.3 Use of Prototyping and 4GL Tools

A Prototype is the model of software to be built. A Prototype helps in the establishment of requirements more clearly. They may be constructed to simulate the business functionality of the system, its scope of coverage, ease of use and suitability to the organisation's way of working, etc. Once the initial vagueness about the users' expectations and functionality of the systems has been cleared by prototyping, a more formal systems analysis and design can refine the prototype further. Thus, by using prototyping as a complement to the use of a methodology, the advantages of both approaches can be retained.

3.6.4 Relational Databases

The advent of relational database technology opened up the use of databases directly by the end users with minimal programming skills. Today there are many RDBMS available under different groups of hardware platforms. Some of the databases and the hardware platforms that are available are tabulated below:

H/W Group	RDBMS
PCs, Minis, Mainframes	Focus, Ingres, Oracle, Sybase, Informix, Unify, etc.

From the above, it is clear that most of the databases available do not meet the first requirement, availability across divergent hardware platforms. Only Oracle and Ingres are available under different platforms, i.e., on PCs, minis and mainframes. Most of the other databases have interface to import data from and export data to different databases on other hardware platforms. But such interfaces do not give optimum performance. As a result, developers get bogged down by performance issues. Interfaces available with other databases are not user-oriented, and hence they are not very user-friendly.

Two databases, viz., Oracle and Ingres, can run across different platforms. They can also transfer data across various hardware platforms without any conversions of programs.

These databases have industry standard SQL and reportwriter which are 4GL tools. Screen-oriented development tools for painting entry screens and menus is an inherent feature of both Oracle and Ingres. This not only facilitates faster development, but also enhances Professional productivity.

Oracle and Ingress have a query optimizer. The main function of a query optimizer is to determine automatically the fastest method in which a database request can be handled. As a result of this, programmers and end-users do not need any additional training to obtain good RDBMS performance. These databases have servers or data managers which minimize both memory and CPU resource utilization. This ensures high performance during transaction processing.

None of the above mentioned databases have the feature of compound document handling. RDBMS have always had robust tools for fixed-length alphanumeric data, and tracking of applications. However, they have not been able to handle unstructured text. Their only mechanism for searching words and phrases in a text field is through sequential string match over the entire database. Some databases have text retrieval applications on top of their RDBMS, but their server programs are not designed for large text transaction. As a result, they suffer from poor response time and lack of text navigation features.

Thus, a compound document having a collection of separate data objects like text, graphics, images, logical structures, layout structures, voice and annotations that can be edited, formatted or otherwise processed as a whole cannot be stored and retrieved by a RDBMS.

The databases of the future should not only be able to handle different objects (Object Oriented Databases) but should also have the following additional features to achieve, if not a paperless office at least a less-paper office.

- Accept document from disparate sources
- Handle several document structures
- Provide complete document identification
- Thesaurus-based concept searching for text
- Stopword and title control while indexing
- Context searching for text
- Device independent searching
- Soundex and plural control
- Reproduction of stored voice after successful search
- Window based interface for end-users.

3.6.5 Object Oriented Programming

As the sophistication and capabilities of new computer hardware have grown by leaps and bounds, it has become evident that new software development techniques are needed. Users anxiously wait for more software that harnesses the capabilities of sophisticated hardware such as the Macintosh II and the PS/2 family of computers. In addition, users' expectations of software quality have increased. They are no longer satisfied with applications that are either not user-friendly or having numerous bugs which must be worked around.

A recent approach to software development, called object-oriented programming, attempts to completely alter traditional software development methods, and many computer professionals believe it has the potential to satisfy some of the demands for more sophisticated software. Object-oriented programming requires a new way of programming, one more closely related to how we actually think.

We typically regard the computer as a machine and data as the raw material that the computer processes. The programmer is the person who controls the machine. The program lists all the steps that the computer must take to obtain the needed output from the input. However, an object-oriented program defines the data and those operations that can act on that data as one unit called as object. The object is thought of as an actor with a specific set of skills. The programmer is the director of the show. The programmer no longer has to tell each actor exactly what steps to take but instead simply explains the ultimate task to be performed.

Critical to understanding object-oriented programming is the concept of inheritance. Objects can be defined and then can be used to build a hierarchy of descendant objects, each of which inherits access to methods used by the ancestors' objects. Objects can be reused. Even when a new object is needed, an old one can be modified to meet the new needs. The new object inherits the characteristics of the old object.

For example, a horse is a subclass of mammals. It inherits the characteristics of a mammal. However, it also has characteristics that distinguish it from other mammals, such as its size and shape, the way it moves, and the kinds of sounds it makes. When a programmer creates a new object, it is necessary only to add its new features: the inherited ones are already there.

To see how this makes the programmer's job easier. Assume you were writing a space-war game. Both sides have space ships but of slightly different types. In addition, each side has not-fighting ships, such as space shuttles and cargo barges. The object ship has certain characteristics: X-Y coordinates, shields, warp speeds, and loyalty. The object fighting ship plus photon torpedoes. The object shuttlecraft has everything except shield and warp speeds.

Many programmers agree that object-oriented programming can greatly reduce the time needed to implement new software. In addition, because new software builds heavily on existing objects, the code is more likely to be reusable and error-free. Several object-oriented languages are currently available. The first commercially available language was Xerox's Smalltalk. Other languages include C++ (an object-oriented version of C), and Borland's Turbo Pascal. The future will determine whether the full potential of object-oriented programming is as great as many computer professionals believe. If it is, then we shall see tremendous improvements in the peed of software development and the quality of the final product.

3.6.6 Graphical User Interfaces

Graphical user interfaces (GUIs) offer a standard look and feel to application, thus reducing development and learning time. A GUI is an application environment that can work with graphical objects. Microsoft Windows, a typical example, has the following components:

- Menus (pull down, pop up, etc.)
- Icons (for identifying applications and resources).
- Tiled windows (for views of multiple programs of data or for multiple views of single program of data block).
- Dialog boxes for selecting files, options, and settings; when an option is selected, the one previously selected is turned off.
- Checklists from which the user can make multiple selections.
- Support for a pointing device, typically a mouse (especially to select and drag screen elements).
- Scroll bars along the edges of windows to show the relative position of the contents (such as the end or beginning of the text) or to move to a different position (such as another part of a spreadsheet).

A GUI enforces consistency by restricting developers – they must support features for the program to run under the GUI. In addition, suggestions from the GUI's creators (such as the arrangements of menu options) often become de facto standards for applications.

We have described how consistency simplifies the learning of a new application. One benefit of a GUI is that the user can learn a second application faster because he or she is familiar with the environment. Note that the benefit comes with succeeding applications.

Consistency and familiarity help produce shorter learning curves. Users generally prefer the interface style they know, whether it be Macintosh, Microsoft Windows, or Lotus 1-2-3. The consistency offered by a GUI trades on the user's familiarity with the environment.

Drawing and CAD programs are best suited to GUIs since, by their nature, they manipulate objects, lines and curves, and fill closed areas with colour. For database programs such manipulation is not as useful. However, they can use GUI's effectively to:

- Specify data field when setting up reports
- Select sort keys
- Transfer data to or form other applications (such as a spreadsheet).

The last point is particularly important. Database application often must transfer data to or from spreadsheets, word processor, desktop publishing programs business or presentation graphics programs, statistics programs, or project management software. GUI generally have data exchange features, such as Microsoft Window's Dynamic Data Exchange (DDE), to handle such transfers.

Hewlett-Packard's New Wave extends the direct manipulation approach. The user can, for example, drag file icons to the printer without loading the applications that crated them. New-Wave also handles the merging of application data with links that are transparent to the user. To add to the general confusion, note that New wave is a GUI that runs under another GUI (Microsoft Windows).

Another benefit of a GUI is that it lets you see the final product before you print it. What You See Is What You Get (WYSIWYG) is a feature essential to desktop publishing and drawing application, and useful in database applications.

However, there are drawbacks associated with using GUI. The costs include the expense of graphics cards, pointing devices (such as mice), and extra memory. Because GUIs run in graphics mode, screen refresh is usually slower as well. If speed is important, then GUI's consistency may not be sufficient compensation.

☞ Check Your Progress 1

1) CASE stands for _____.

2) A _____ is the model of software to be built.

3.7 USING THE METHODOLOGY

Very often, as compared to organisations with systems departments that cater only to in-house development requirements, many IT consultancy organisations have some kind of in-house standards and procedures in planning, designing and developing their clients' systems. The extent to which these methods, are formal may vary from organisation to organisation and be influenced by the environment in which the consultant operates (e.g., more formal, if addressing the international market or if the consultant has some business tie-up with a multinational corporation).

The scope and complexity of different methodologies vary considerably depending on the objectives that the methodology seeks to achieve and the tools and techniques they prescribe.

Some of the areas in which current day methodologies are deficient are in their support to the testing and maintenance phases. The usual response to this is to adapt them or to dovetail them with in-house standards governing these areas. Sometimes, it is said that following methodology too closely (with its insistence on formalizing the whole process) may lead to the development of systems of Yesterday. Also, expecting

miracles on Day 1 of introducing the methodology and believing in it as a panacea may lead to disillusionment.

As the saying goes, "A methodology does not replace a good System Analyst, it only makes a good Systems Analyst better". The ultimate aim of an organisation is still to build better systems, not to follow an excellent methodology.

3.8 CHOOSING THE RIGHT METHODOLOGY

Given the above scenario in the methodology market, how does one go about evaluating the various options, to decide which methodology is suitable for one's organisation? Well, there are no readymade answers, and there is obviously no single 'best solution' that is suitable for all organisations and for all its projects. Depending on the problems faced by individual organisations and the setting in which they do business, the objectives in adopting a methodology may vary. Thus, what seems to work very well in one place may introduce more chaos in another. However, the following questions may be asked about each methodology before deciding on adopting it as a standard.

a) Scope and level of detail

- Is it applicable to various types of systems that your organisation builds (e.g., transaction-oriented, process-oriented, real time, small vs. large)?
- What is its scope and what are its boundaries? Does it cover business planning, IT strategy, feasibility, analysis, design and maintenance phases? In what detail does it cover these?
- Does it facilitate cross-reference between products of various techniques (e.g., DFDs with ERDs).

b) Integration with other tools

- Does it have enough flexibility to integrate prototyping
- What is the data dictionary scheme it supports
- What is the documentation standard it proposes? Is it easily created, referred and maintained?
- What output of platforms does it provide to facilitate project management?
- What does it recommend as quality assurance mechanism?

c) Ease of learning and use

- What level of training does it require for use?
- What is the support available from the vendor for training and consultancy while practicing the methodology?
- What is the feedback from other users? Are there any user groups active?

3.9 IMPLEMENTING A METHODOLOLGY

Recognizing the need for 'good' methodology and appreciating its benefits may just be the first step, and a lot of hard work still remains to be done before a methodology can be successfully introduced in an organisation.

The following are some of the important steps:

- The type of applications that are developed by your organization (transaction processing, decision support, on-line, batch, etc.)
- Understand the problems you face more often and which ones are of more concern to you.
- Look around the methodology market and select the one (s) which are right for you.
- Involve senior management in the whole process and educate the personnel directly or indirectly involved (systems staff and end-users).
- Train systems staff on the intricacies of the methodology.
- Apply the methodology on a pilot project (small, non-critical, low priority application system) to get the feel of it.
- Ensure you get feedback on its effectiveness.

The usual resistance to the introduction of a methodology is that it is looked at as 'Old wine in a new bottle, and that it takes away the freedom and creativity of doing one's work..

Tools for software development

Just as many tools exist for building a house, many tools are available for creating or writing software. These tools comprise different types of programming languages, each of which consists of a number of different commands that are used to describe the type of processing to be done, such as multiplying two numbers together. Software development tools can best be categorized as falling into one of five generations of programming languages. The languages in each successive generation represent an improvement over those of the earlier generation – just as the electric saw was an improvement over the manual one. Languages of later generations are easier to learn than earlier ones, and they can produce results (software) more quickly and more reliably. But, just as a builder might need to use a manual occasionally to cut a tricky corner, professional programmers still need to use early generation languages (except machine language, which we'll explain shortly, to create software. Each of the five language generations will be described in detail in this unit).

Compared with later generations, the early generation of programming languages (first, second, and third) require the use of more complex vocabulary and syntax to write software; they are, therefore, used primarily by computer professionals. The term syntax refers to the precise rules and patterns required for the formation of the programming language sentences, or statements that tell the computer what to do and how to do it. Programmers must use a language's syntax – just as you would use the rules of German, not French, grammar to communicate in German – to write a program in that language. As efficient software development tools are available, programmers do not develop software using machine language any more, and few use assembly language, except for programs with special processing requirements. However, third generation languages are still in wide use today.

Fourth generation languages still require the user to employ a specific syntax. But, the syntax is easy to learn. In fact, fourth-generation programming languages are so much easier to use than those in prior generations that the non-computer professional can create software after a short period of training.

Natural Languages enabled processing languages will constitute the fifth generation of languages. With this type of language, the user will be able to specify processing procedures using statements similar to idiomatic human speech-simple statements in English (or French, German, Japanese, and so on). The use of natural language may not require the user to learn a specific syntax.

In addition to the five generations of programming languages, some microcomputer software packages (such as electronic spreadsheet and database management software) are widely used for creating software. Although these packages generally cannot be categorized into one of the five generations, many people consider some of the database management systems software used on microcomputers, such as dBASE IV, to fall into the fourth-generation category.

3.10 WHICH TOOLS ARE YOU MOST LIKELY TO USE?

The decision about which software development tool to use depends on what processing procedures you need to perform. Developing software is like building a house: The work will go much faster if you have a plan and the right tools. However, the tools have little value if you do not know how to use them; consequently, one of the most important steps towards effective and efficient software development is the selection of the right development tool.

As most of the Applications that exist in the market today are developed using third Generation Programming Language, the software that you buy off the shelf of a computer store has been created by a computer specialist using one of these languages. Also, computer specialists need to know how to use these languages in order to update, or maintain, this existing software to accommodate new processing and output requirements.

For the user who is not a computer specialist, the most popular tools for developing software will be the fourth-generation programming language and existing off the shelf software packages such as electronic spreadsheets and database management systems software, because one does not have to be an experienced computer professional to use them. The user who is working with these tools can create specialized software applications, such as keeping track of a company's expense by department (a good application for a spreadsheet package), or maintaining a comprehensive customer file used in a clothing store for billing, marketing, and checking customer credit status (a good application for a database package).

☞ Check Your Progress 2

1) Natural Languages enabled processing languages will constitute the _____ generation of languages.

2) For a user who does not have sufficient expertise in computers, the best generation of languages to use are that of _____ generation.

3.11 CURRENT GENERATION OF SOFTWARE DEVELOPMENT TOOLS

Over the past 40 years, the programming languages used to develop software have been steadily improving in terms of ease of use, the time it takes to develop software, and the reliability of the finished product. Here we describe the major characteristics

of current generation of languages, or software development tools, and pay special attention to the tools that may be useful in the business environment.

3.11.1 Fourth Generation Languages

Also known as very-high level languages, fourth generation languages (4GLs) are as yet difficult to define. Sometimes, these languages are tied to a software package produced by the vendor, such as a database management system. Basically, 4GLs are easier for programmers, and user to handle than third-generation languages. Fourth-generation languages are non-procedural languages. They allow programmers and user to specify what the computer is supposed to do without having to specify how the computer is supposed to , which, as you recall, must be done with third-generation, high-level (procedural) languages. Consequently, a fourth-generation language need to achieve the same result. Because they are so much easier to use than third-generation languages, fourth-generation languages allow user, or non-computer processional, to develop software. It is likely that, in the business environment, you will at some time use a fourth generation languages. Five basic types of language tools fall into fourth-generation category:

1) Query language,

2) Report generators,

3) Application generators,

4) Decision support systems,

5) Some microcomputer applications software.

Query languages allow the user to ask questions about, or retrieve information from, database file by forming requests in normal human-language statements (such as English). Query languages do have a specific grammar, vocabulary, and syntax that must be mastered (like third-generation languages), but this is usually a simple task for both user and programmers. For example, a manager in charge of inventory may key in the following questions for a database.

How many items in inventory have a quantity-on-hand that's less than the reorder point?

The query language will do the following to retrieve the information:

1) Copy the data for items with quantity-on-hand less than the reorder point into a temporary location in main memory;

2) Sort the data into order by inventory number;

3) Present the information on the video display unit (or printer).

The manager now has the information necessary to proceed with recording certain low-stock items. The important thing to note is that the management did not have to specify how to get the job done, only what is needed to be done. In other words, in our example, the user needed only to specify the questions, and the system automatically performed each of the three steps listed above.

Some query languages also allow the user to add data to and modify database files, which is identical to what database management systems software allows you to do. The difference between the definitions for query language and for database management systems software is so light that most people consider the definitions to be the same.

Report generators are similar to query languages in that they allow users to ask questions of a database and retrieve information from it for a report (the output). However, in the case of a report generator, the user is unable to alter the contents of

the database file. And with a report generator, the user has much greater control over how this output should look or the user can create his or her own customized output reports using special report-generator command instructions. (Ordinary users may need the help of a computer specialist to use a report generator).

In most reports, the user requires that a total or totals of one or more groups of numbers appear at the bottom. And, if more than one category of information is to be included in the report, the user usually wants subtotals to appear for each category. In the case of a third-generation language, the number of instructions necessary to create totals is about 10 times the number needed in a fourth-generation language because the programmer needs to specify not only what to total but how to total and where to place the total. Report generators have many built-in assumptions that relieve the user of having to make such tedious decisions.

Applications generators, as opposed to query languages and report generators which allow the user to specify only output-related processing tasks (and some input-related tasks, in the case of query languages), allow the user to reduce the time it takes to design an entire-software application that accepts input, ensures data has been input accurately, performs complex calculations and processing logic, and output information in the form of reports. The user keys the specifications for what the program is supposed to do into a computer readable form. The resulting specification file is input to the applications generator, which determine show to perform the tasks which then produces the necessary instructions for software program. For example, a user like yourself could use an applications generator to design payroll runs – to calculate each employee's pay for a certain period and to output printed cheques. Again, as with query languages and report generators, the user does not have to specify how to get the processing tasks performed.

Decision-support systems combine special interactive computer programs and some special hardware to allow high-level managers to bring data and information together from different sources and manipulate it in new ways – to make projections, do what it analyses, and make long-term planning decisions.

Some microcomputer applications software can also be used to create specialized applications – in other words, to create new software. Microcomputer software package that all into this category include many spreadsheet programs (such as Lotus 1-2-3), database managers (such has dBASE IV), and integrated packages (such as Symphony). For example, in a business, without computers, with respect to accounts receivable (to penalize people with overdue account balances), someone has to manually calculate how many days have passed between the invoice date and the current date and then calculate the appropriate penalty based on the balance due. This can take hours of work. However, with an electronic spreadsheet package, in less than half an hour the user can create an application that will calculate accounts receivables automatically. And the application can be used over and over again.

Another example of microcomputer software that is used to create new programs is HyperCard for the Macintosh-created by Bill Atkinson of Apple. In general, this package is a database management program that allows users to store, organize, and manipulate text and graphics; but it is also programmable that uses a new programming language called Hyper Talk to allows the ordinary user to create customized software by following the authoring instructions that come with the package.

3.11.2 Fifth Generation Languages

Natural languages represent the next step in the development of programming languages that fall in the category of fifth-generation languages. Natural language is similar to query languages, with one difference: It eliminates the need for the user or programmer to learn a specific vocabulary, grammar or syntax. The text of a natural-language statement very closely resembles human speech. In fact, one could word a

statement in sever ways – perhaps even misspelling some words or changing the order of the words – and – yet get the same result. Natural language takes the user one step further away from having to deal directly and in detail with computer hardware and software. These languages are also designed to make computer smarter 4GLs or that is, to simulate the human learning process.

3.12 4GLs OR FOURTH GENERATION LANGUAGES

Down the road of computer history, one sees the evolution of computer technology in terms of both hardware and software. It can be traced back to the times of the first generation computer which used vacuum tubes as basic components of internal circuits. Then came the era of second generation computers followed by the era of transistors. Since then a subsequent improvement in the basic design using integrated circuits and then using very large scale integrated circuits led to what was termed as the third and fourth generations of computers. Computers of the fifth generation have also emerged. The fifth generation of computers are those which emulate artificial intelligence that resembles human intelligence. This generation of computers represents a leap into knowledge processing compared to data and numerical processing carried out in the computers of all the previous generations.

Scanning the development phase, in the field of software we see that the first generation software was very near to machine language coding. Since then, the following generation of languages have attempted to ease the effort which goes into programming. Second generation software was using a command language, e.g., the job control language (JCL) used in IBM 360 computers.

Third generation languages which are very commonly used include C, COBOL, Pascal and PL/1. We have entered into the era of fourth generation languages with languages like Focus, Ramis and Linc.

3.12.1 What is 4GL?

Computer hardware has evolved through various generations as the vacuum tube gave way to transistors, then integrated circuits and subsequently to very large scale integrated circuits. Computer languages have also kept pace with this trend and have evolved over a period of time, from machine on first generation languages, featuring intricate combinations of 0s and 1s through assembly level languages (second generation) and the third generation COBOL, BASIC, etc., presently to the 4GLs.

Most application software that is available these days is the one written in third generation languages, like COBOL, BASIC and C. Till data, these languages, also known as higher level languages, have been used to solve any application demand whether suited for it or not. This results in a lot of necessary code which drains a great deal of time in programming activity and also increases the response time, thereby causing a dip in the efficiency of the computer. So, what was needed was a computer language which could do everything that a third generation language does but with much less effort. Hence what emerged was a fourth generation language.

A 4GL can be defined as a very high level computer language that enables rapid development of application, sometimes without the help of information systems (IS) professional, aiming to improve productivity in computer systems development and use.

A survey conducted to assess the gains of a 4 GL showed that it required one-tenth of the time and effort to develop software using a 4GL. A 4 GL is more of an application building language and has all encompassing syntax for every aspect of application building.

Trained manpower for software development is a scarce resources all over the world while the hardware costs are dropping, the professional staff are becoming more expensive and harder to recruit and retain. Economic law dictates that there should be an effort towards replacing the more expensive resources by the less expensive ones. 4 GLs provide a means to do so. 4GL significantly affects two major factors, i.e., effort and time.

Comparatively less effort is needed in designing applications using 4GL as these are generally very user-programmer-friendly. The programs written in 4GL have to be specified with what is required of the task and in what particular sequence it needs to be done. It requires much less expertise to write down the code compared to what is needed to program in a third generation language. Subsequently, much less effort is needed to debug and modify the programs.

Not only does 4GL let you build applications faster, it lets you run them faster too. These speeds have been achieved through the combination of automatic indexing, concise instruction set abbreviated instruction, clear and well defined condition-all together lead to improved programming productivity. Enhanced query optimizers and report generators go a long way in faster access to the information for the people who depend on it the most.

3.12.2 End User Computing

Users themselves can get information out of computerized databases. This ensures that information systems can respond to end-user as per their needs. In that sense the system becomes demand driven and not supply driven.

3.12.3 Prototyping

Usually a considerable amount of time is spent in arriving at the correct functional specifications. Over the years, a number of methodologies have been developed to help this process. 4GLs offer prototyping as a technique to reduce the time spent in the process. Application development in a prototyping environment proceeds as follows:

- Information system professionals quickly arrive at a prototype of the system.
- Users and information system professionals together review prototype and make changes till the prototype is correct.
- Informational system professional optimizes the database design for programs.
- The critical programs are rewritten for better throughout.

3.12.4 Non-Procedural Languages

A 4GL programmer writes a program specifying what needs to be done and the appropriate procedure to accomplish the task. This shift towards non-procedural programming de-skills the expertise required to write programs and also makes it simpler to modify the existing system.

3.13 CONSIDERATIONS IN APPLICATIONS DEVELOPMENT

Application development is still a bottleneck in most organisations' effective use of, and satisfaction with, the computer. For the purpose of discussion, we are including all processes involved from conception of the application through its development in its entirety. The considerations that apply in developing a suitable application and the problems associated with each consideration are given below.

The program must enable the user to do whatever he is doing currently as well as whatever may come up in the future which he has to do preferably.

1) Most users expect the programs to be ready in a short time.

2) Programs must be error-free.

3) Programs must be modified at short notice to take care of missed or new requirements.

4) Turnover in personnel or computer systems must not interrupt the running of programs.

5) Costs must be justifiable.

6) Some PC-based applications are meant for use by a senior manager or director of the company and are expected to be developed by a more or less trial-and-modification approach.

3.13.1 Problems in Application Development

1) This simple requirements is unfortunately not simple to accomplish as it necessitates the programmer to master the application before developing it. This requires time from the user and programmer, dialogue between the two, good communication skills on the part of both, and an effective recording of the understanding.

2) For the system to be better than the current one, innovative thinking is required and like most innovative ideas, they require considerable time for experimenting and check-out.

3) Trying to develop a system in a short time generally leads to tension and attempts at short cuts which affect the quality which, in turn, affects the schedule.

4) Programs typically deal with hundreds of abstract logical processes and to guarantee a 100 per cent error-free performance is generally impossible. And, yet even a single error seems shocking. Probably, 60 per cent of the programming effort goes in trapping the 5 per cent errors and misunderstandings that enter the system. Testing programs is still more a skill than a science and accuracy is a function of the skill of the programmer, the complexity of the system, and the effort spent in testing.

5) Modifying programs is the most error-prone activity and gets tested the least because it is required soon, and testing the program fully takes a lot of time. And often, the part that is not modified malfunctions because of unanticipated interactions.

6) Programs being complex, to train someone else to take over a program requires substantial efforts, and with other schedules in the pipeline, as well as maintenance requirements, it is generally done only when a notice of termination of transfer is received. When someone leaves without adequate notice, it can be a disaster. Change of computer systems to incompatible languages is a huge project and must only be done with a significant budget, big expected gains, and a strong heart.

7) This is, of course, the bottomline and with the other problems involved, is a real challenge.

8) These MIS systems do not justify a rigorous Systems analysis and Design approach and yet the quality is expected to be very high.

3.13.2 How 4 GLs Help to Solve Problems?

For the purpose of comparison, LINC or MAPPER is treated as a representative 4GL and COBOL as a representative 3GL. The following are the distinctive features of a 4GL:

1) They are much easier to learn and use.

2) They provide more powerful features, so fewer commands are required to accomplish the task at hand.

3) They provide convenient feedback on syntactic mistakes and enable the user to correct the same and continue the program.

4) They are being improved at a much faster rate than 3GLs.

The above features have, in a sense, introduced a revolutionary change in the programming scene. Some of the changes are explained below:

1) Many young professionals, who are not programmers, feel and become competent enough to develop programs for their areas of interest. Due to their expertise in their applications, their programs meet their requirements. As they represent both the user and the programmers, they are more user oriented and more cost-less and benefit-more oriented than a typical programmer. Thus, their programs are simpler to the necessary point, and easier to develop. This contributes to reducing many of the problems associated with applications.

2) Programmers spend less time in writing and testing programs due to the brevity of the commands. This has improved programmer productivity by about 2-3 times.

3) Ad-hoc information can be, comparatively, easily provided.

4) It is easier to develop models of the systems. They take a fraction of the effort to develop and it helps to clarify the understanding of the application and thus serves a purpose similar to an architect's blueprint of a house that is to become a home.

5) The on-going improvement in 4GLs promise increasing productivity of programmers and computers.

3.13.3 Limitations of 4GLs

Is it necessary that the organisation using a 4GL for its database development will show a dramatic increase in productivity? Will the use of some 4GLs bring about a significant improvement in the organisation's performance? The answer might be a No. This is because our industry, overwhelmed by very conspicuous advantages, is offering 4GLs as a solution to all possible problems. Unfortunately, people are overlooking that there is a substantial gap between where they are and where they want to be.

Let us view why 4GLs are not delivering the goods even though they have some very obvious benefits.

Probably the main reason for this is the not-so-good performance of many products. Thus, might be because these products provide programmer productivity gains, but leave little scope for the designer or the analyst.

The approval of a 3GL in development was concentrated on the paper intensive technique which required a lot of time, effort, expertise and experience. 4GLs, by contrast, support prototyping (which are working models of the system and can be changed quickly and easily) and methods of modifications which enable the user to be

directly involved in the process of development – as the end user's requests can be accommodated and the change reflected in a very short span of time.

Secondly, the database management system supporting the 4GL might not be sound. For example, if a relational DBMS is being implemented, then it is required that the non-redundant tables should be used – which is not very often the case.

Another reason why the vendor claims of huge productivity gains from programming are not met is that not all the products claiming to be 4GL qualify to fall under this category. They can be carrying out one or more functions of a 4GL. To list only a few, these products might be just query processors running against some file management system. They might be high level languages teamed with a DBMS, or they just might be COBOL code generators.

Hence to really get value for one's money spent, understanding of the differences and the gap between the old and new generation technologies is required.

Possibly, the biggest problem in using 4GLs is that experienced 3GL programmers have to invest considerable effort (about a few man-months) to master a new language. Also, since the installed programs in COBOL, BASIC, etc., are very many, and converting them to 4GLS is generally a prohibitively expensive effort, the new 4GL programs must also provide a mechanism for a full two-way transfer between 3GL and 4GL systems.

3.13.4 Impact of 4GLs

The following are some areas wherein the impact of 4th generation languages (4GLs) deserve attention of management.

- Productivity and cost of software development.
- Restructuring of the systems development process.
- Increased emphasis on decision support system (DSS) and end-user computing (EUC).
- Changes in the roles of users and systems professionals.

Let us study each in some detail.

Software Development

One major reason for the development of 4GLS is productivity. The 10-to-1 gain is now becoming visible, and though we see applications getting developed for implementation which a tenth of the man hours that it took with, for instance COBOL, it does not mean that cost has become one-tenth. The extra hardware resources required by 4GLs (more CPU cycles, additional memory, etc.) contributes towards some increase in cost, but in spite of this, 4GL solutions cost much less. And in the years to come, hardware cost will continue to plunge downwards while cost of each technical man-hour that is used for the development of software will move upwards rapidly.

Productivity increase takes place because of two characteristics of 4GLs. First, that every man hour of programming generates much more lines (COBOL equivalent) of program, and second, that the level of skills required to write programs in 4GLs is typically lower than that required to write programs in 3GLs like COBOL. The management implications are obvious. On the one hand, lower development cost means that more and more applications become cost effective for computerization and, on the other hand, there is a faster turnaround of applications because of which computerization can progress more rapidly.

System Development Process

It begins with the generation of idea like "why don't we evaluate the possibility of computerizing the maintenance planning activity?" This type of thinking termed Conception usually starts in the minds of senior managers, and leads to a process of evaluation in the form of a brief feasibility study. For this purpose, a group of staff evaluate the idea from two viewpoints – the technical feasibility and economic viability. This stage which may be termed Initiation is required because it is important to evaluate each attempt at developing systems. Cost can be prohibitive and application development skills are scarce. So, rigid discipline (of ensuring that each idea is viable) needs to be introduced and no project should begin till this is done.

The analysis phase is carried out by the systems analyst once a clear go-ahead is obtained from the evaluation group. It ends with a very well defined set of deliverables, termed Functional Specification. During the design phase, the system designer converts functional specification into yet another set of highly defined parameters and guidelines that include file design, codification structure and program specification. Programming is the next phase and it all ends with rigorous testing that concludes the development cycle.

It is a widely accepted fact (though not as widely practiced) that to keep things under control and in order, the sequence of phase in the SDLC should be followed very rigidly. This means that the designer should into begin his work till the analyst has finished. And it should be accepted that Analysis is not over till such time as the user has signed and accepted the output formats to the very last detail. This also implies that once the user has committed himself to a set of output formats, it is wise not to seek changes till the system is settled and running smoothly since a good part of the subsequent activity (design, programming and testing) would have to be re-done.

One major reason attributed to the sizeable delays that usually take place in development is that the SDLC is not followed rigidly. The usual tendency is to take short cuts in analysis and then implement changes once the system is almost complete. This plays havoc with time-frames as well as the stability and reliability of the system since changes introduced later can generate additional errors that may go undetected.

The impact of this phenomenon has been quoted often. If those changes that are sought after the system is operational have been identified earlier, through more intensive and thorough analysis in the Systems Analysis phase, then there would be an almost hundred-fold saving of that time. In other words, 100 hours spend in modifying a system that is operational could have been saved for every extra hour spend during analysis to ensure complementness and accuracy.

4GLs provide an environment where it is possible to iteratively and simultaneously carry out analysis, design and programming. This activity called prototyping needs skills as well as expertise and it is appropriate for systems of low to medium complexity. In prototyping, analyst identifies a part of the functional specifications and while he is carrying on with the remaining part of the analysis, some part of the design (the in-built database packages in the 4GLs aid this process immensely) is also made. Programming in any case, is a simpler activity with 4GLs and is carried out concurrently for that part of the system which has gone through design. As a result of this, a small, but perhaps the most important part of the system is set up early and the end user can evaluate it for suitability before the remaining part is worked upon in greater detail. 4GLs provide inherent ease in polishing up the prototype using the embedded database coupled with powerful and easy-to-use interfaces.

Prototyping is no child's play, though today's software tools need major improvements to make this an easy process. Nonetheless, the way in which 4GLs are evolving points to the distinct possibility of this begin the dominant manner in which systems will be developed in the future.

Increased emphasis of DSS and EUC

While productivity increase is a major objective of 4GLs, an equally important expectation from 4GLs is their ease of use. On this count, all 4GLs are not equal but some even provide features that make it possible for end-users to create their own reports with a very short period of training.

This is accomplished through the provision of end-user interfaces and program generators. In *Focus,* there is the revolutionary TALK technology that enables users to create complex reports as well as graphical outputs by interacting with the system through only the ARROW and ENTER keys. The process involves a series of interactions, where the computer screen shows a set of options and the users moves the cursor to the required option using the arrow key up, down or to the side-and then confirms the option by pressing the ENTER key. This throws up the next screen, and the process is repeated as the user moves the cursor and chooses an option.

This dialogue continues till the user has input the desired choices, at the end of which the report is generated.

This new found ability of the end-user to develop his own reports open up new vistas in computerization. Once the database is in place (Information System professionals would have played some role in the design of the database), end-user is free to seek information and schedule that he wants it.

Ad hoc information retrieval is the first step in the process of development of more and more sophisticated Decision Support Systems. It is also the most commonly used application by top level managers. Gradually, we see these of mathematical models and statistical techniques.

Leading 4GLs provide such tools in the form of building blocks and even these tools come with easy-to-use interfaces.

Relational database structure coupled with versatile modeling tools and front-ended by powerful graphics as well easy-to-use reporting functions-all of which are constituents of the more popular 4GLs-are making end-user computing and Decision support Systems a reality that can be actualized by end-users themselves.

What it means from the management viewpoint is that, now, it is becoming possible to allocate part of the software development responsibility to user departments directly.

Changing Roles

First, it becomes mandatory to train end-users adequately to cope with the new responsibility.

Secondly, it is possible to draw up more ambitious computerization plans since a much larger number of people are directly working on the development of new systems.

Thirdly, top management needs to change he role of the IS professional from a developer of a developer of application to one of consultant to a large number of developers, i.e., end-users.

And finally, management needs to put a comprehensive set of policies, procedures, standards, and guidelines (to be developed and monitored jointly with IS staff) in place that will enable end-users to develop profitable applications with standard software tools, using rigid consistent document standards across the organization.

3.13.5 What to Look for in a 4GL?

The issues which go into making of a 4GL need to be carefully equated to have the greatest impact on the performance, resource usage and productivity. It is the 4GL design that extends or limits the ultimate degree of success in using it. Hence before

entering into functional evaluation, one must understand how the 4GL is built and designed. Portability is another aspect to ponder over. This is because, it is quite likely that some time or the other, a new version of the operating system might be installed or some change in the hardware configuration may be brought about. Hence questions about the present configuration and case of migration to the new environment should be asked.

LINC

LINC is an example of a 4GL that is being used at more than 3000 sites around the world. Line enables you to design, develop, modify and generate on-line applications systems. It consists of a definition language and an interactive computer which checks the validity of a coding as you input it and creates the program for the system. It allows the definition of the problem through a brief series of business-oriented, English-like statements which help minimize the errors and misinterpretations usually associated with complex programming languages. Errors in the code are highlighted and the corresponding explanation of the errors can be listed by giving an 'ERROR' command. This permits rapid development of the software code.

The Linc-interactive computer is a menu-assisted system which operates thorough an activities menu. The menu displays the activity modes which are used to create Linc systems, reports, and networks. It is possible to access any one of them in any order. Each activity mode consists of one or more input screens through which one can define the appropriate part of the Linc system being developed, e.g, reports are defined through Report mode and networks through the Network mode.

Linc was developed for the business functions and hence its basic structure is business activity compatibility. A linc system definition consists of three basic parts: components, events and profiles.

Events are business transactions that occur with this fixed data, e.g., receipt of goods, sales, cash receipts and so on.

Profiles are used to specify the various ways in which the components and events in a system are to be combined, viewed and accessed. For example, profile of a component which deals in employee information could be a way to access or view the employee information in the ascending order of employee number. It might be in the order of employee name depending on the specific requirements.

Components and events are initially included in the Linc system by keeping the (master file and transaction file) idea of the business world in mind where components simulated a master file and events simulated the transaction file.

Creation: Using the Linc definition language and viewing the business activity in terms of components and events, the logical specification for them can be declared.

Generation: Once the specifications are defined through the LDL (Line Definition Language) the generation phase begins in which all the applications system

Specifications are established and maintained and the system made ready to use. This is done without any further programming activity. In this, each functional aspect of programming code needed to crate the system is generated automatically by Linc. On this generated system, it is possible to 'add', 'change', 'inquire', and 'delete information that is present in the database.

Reports can be made to run and also a query session can be started.

Reports: Reports are developed by the use of LIRC (Logic and Information Report Computer), which describes the function of each LIRC specification using LDC (Linc Definition Language). LIRC is the reporting aspect of the Linc system. Once a Linc information system has been designed, LIRC may be used to design the reports required for accessing and presenting the information recorded in the system.

Some Examples

An accountant, who employs computers, developed the accounting system for his company alone in just two months (using dBASE). He and his company are very happy with the system in terms of its productivity and reliability. Typically, an experienced non-accountant programmer would have taken over a man-year to develop the same application in COBOL, and possibly about 5 months in dBASE. Here, four productivity factors got compounded; dBASE is more productive than COBOL; an accountant programmer is more productive than a non-accountant programmer; user programmer is more productive that a non user programmer; and dBASE enabled the other 3 productivity factors to be more practical. A programmer chartered accountant firm routinely turns out customized applications in almost the same cost and time frame it takes to implement a packaged software. Probably, the reasons are: they are very competent, they quickly grasp user requirements, they program in the dBASE language, and they religiously try to keep their systems as simple as the can. The same arguments may not hold true for future as more and more sophisticated software is developed.

Prospects for the future

Probably, more programs are currently being developed in 4GLs than in 3GLs. This trend is expected to grow. As more and more manages are expected to use computers as they use their calculators and diaries, they will turn to 4GLs for meeting their information requirements. Also, improvements in 4GLS promise more power and ease to the users.

There are several hundred products in the market that can claim to be 4GLs. The production cover program generators and virtually anything that encourages an end-user to produce applications without resorting to the use of conventional programming language.

☞ Check Your Progress 3

1) DSS stands for _____.

2) EUC stands for _____.

3.14 SUMMARY

4GLs ensure easier and faster development of applications. Broadly, they cut down the number of lines of program code required and they provide a simplified approach to the design of program, such that (in theory) the end-user is able to do the programming work for a particular task s/he requires.

4GLs have not, however, had a acceptance that the promotional hype would seem to warrant.

- They cater for the production of only new applications.
- Most data processing departments and information centers spend a good deal of their resources on running systems for users but 4GLs do nothing for the operations area.
- 4GLs exemplify the software equation that capability demands more of a computer. 4GLs need a fast computer and a lot of memory.
- Programs written in 4GLs cannot mate well with existing software.
- There are no universally agreed upon standards.
- 4GLs may be relatively easy to learn, but the basic principles of good programming still apply. The need-user may not have the training or the

professional background to follow them. That all takes skill; it can also take a good deal of time.

- The availability of 4GLs has encouraged end-users to utilize them in developing relatively trivial applications rather than the more sophisticated tasks for which they were intended. 4GLs can produce large clumsy program for such small jobs.

3.15 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Computer Aided Software Engineering

2) Prototype

Check Your Progress 2

1) Fifth

2) Fourth

Check Your Progress 3

1) Decision Support System

2) End User Computing

3.16 FURTHER READINGS

1. *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.
2. *Software Engineering, Sixth Edition, 2001*, Ian Sommerville; Pearson Education.

Reference Websites

- **http://www.rspa.com**